Universidad de Valladolid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Diseño e implementación del servidor de una aplicación socio-semántica de anotación de árboles

---

Autor:
Cristina Mayo Sarmiento

Tutores:
Dr. Guillermo Vega Gorgojo y Dr. Miguel Luis Bote Lorenzo

Valladolid, 25 de septiembre de 2020

Título:	**Diseño e implementación del servidor de una aplicación socio-semántica de anotación de árboles**
Autor:	**Cristina Mayo Sarmiento**
Tutores:	**Dr. Guillermo Vega Gorgojo y Dr. Miguel Luis Bote Lorenzo**
Departamento:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

Presidente:	**Dr. Manuel Rodríguez Cayetano**
Secretario:	**Dr. Javier Manuel Aguiar Pérez**
Vocal:	**Dr. Juan Pablo de Castro Fernández**

Fecha:	**25 de septiembre de 2020**
Calificación:	

RESUMEN

Los datos forestales existentes en España y en otros países del mundo se recogen de forma oficial y son mantenidos por organismos públicos que los publican como datos abiertos, típicamente en inventarios forestales. A pesar de la gran cantidad de información recolectada, esta suele ser limitada y restringida a datos textuales o cartográficos debido al gran coste que supone a las administraciones públicas el muestreo de las masas arbóreas de un país. Aunque estos datos tienden a ser muy precisos y son de dominio público, su explotación presenta dificultades, especialmente a usuarios no expertos en el dominio forestal. Adicionalmente, la integración de datos resulta compleja a causa del uso de modelos de datos dispares y formatos propietarios.

En este Trabajo Fin de Máster se propone el diseño e implementación del servidor de una aplicación social que permita la anotación semántica de árboles y el etiquetado de especies por cualquier ciudadano, con el objetivo de enriquecer los datos inventariados existentes, permitiendo la creación de datos relativos a nuevos árboles no registrados en los inventarios oficiales. Se ofrece la posibilidad de añadir imágenes de los árboles, etiquetarlas y anotarlas, además de anotar su posición y especie, todo ello de forma colaborativa. También permite sugerencias sobre los datos inventariados, a veces obsoletos. Aborda los inconvenientes relativos a la integración, los modelos de datos y formatos de los datos forestales y su publicación en los inventarios mediante el uso de tecnologías de la Web Semántica, de forma que estos datos estén en formatos estándares, sean procesables por ordenadores y se faciliten las integraciones de distintos tipos y fuentes de datos. Por ejemplo, los datos del Inventario Forestal Nacional (IFN) convertidos a datos abiertos enlazados por el proyecto Cross-Forest se pueden integrar con la aplicación para que, aparte de generar sus propios datos sociales sobre árboles, sirva como punto de la consulta de datos oficiales por cualquier interesado de forma mucho más sencilla.

La aplicación planteada tiene potencial utilidad en dos áreas bien distintas, en el marco de la administración pública, para la gestión más eficiente de parques y jardines y en entornos educativos, como herramienta de consulta y aprendizaje en el etiquetado e identificación de especies de árboles. Este trabajo se centra en el *back-end*, donde destacan las tareas de creación de una ontología propia para el modelo de datos de la aplicación y el estudio y reutilización de otras ontologías existentes con el objetivo de emplear datos abiertos enlazados. Por otro lado, a nivel de *software*, cabe señalar el desarrollo de una API REST compatible con distintos *front-end*, de manera que la complejidad asociada a las tecnologías de la Web Semántica permanezcan ocultas para estos, funcionando como un *proxy* entre los recursos expuestos con representaciones en formatos estándares conocidos y las consultas SPARQL sobre los datos enlazados existentes en el almacén de triplas. El prototipo realizado ha sido diseñado para que sea ampliable en el futuro. Actualmente integra los árboles del IFN de la provincia de Valladolid, si bien, se espera ampliar estos datos en el futuro con los de otras provincias e incluso países, así como ofrecer la posibilidad de añadir y consultar datos forestales técnicos como medidas dendrométricas.

PALABRAS CLAVE

forestal, anotación socio-semántica, API, Web Semántica, Datos Abiertos Enlazados

ABSTRACT

Existing forest data in Spain and in other countries around the world is officially collected and maintained by public bodies that publish it as open data, typically in forest inventories. Despite the large amount of information collected, this is usually limited and restricted to textual or cartographic data due to the high cost to public administrations of sampling the tree stands of a country. Although these data tend to be very precise and are in the public domain, their exploitation presents difficulties, especially for unskilled users in the forestry domain. Additionally, data integration is complex due to the use of disparate data models and proprietary formats.

This Master's Degree proposes the design and implementation of a social application server that allows the semantic annotation of trees and the tagging of species by any citizen, with the aim of enriching existing inventory data, allowing the creation of data relating to new trees not registered in official inventories. The possibility is offered of adding images of the trees, labelling them and annotating them, as well as noting their position and species, all in a collaborative manner. It also allows for suggestions on the sometimes obsolete inventory data. It addresses the drawbacks related to integration, data models and formats of forest data and their publication in the inventories through the use of Semantic Web technologies, so that these data are in standard formats, are readable-machine and facilitate the integration of different types and sources of data. For example, the spanish "Inventario Nacional Forestal" (IFN) data converted to Linked Open Data by the Cross-Forest project can be integrated with the application so that, apart from generating its own social data on trees, it serves as an easily point of reference for anybody interested in the official data consultation.

The application has potential utility in two very different areas, in the framework of public administration, for the more efficient management of parks and gardens and in educational environments, as a tool for consultation and learning in the tagging and identification of tree species. This work focuses on the back-end server, where the tasks of creating a proprietary ontology for the application's data model and the study and reuse of other existing ontologies with the aim of using Linked Open Data are highlighted. On the other hand, at a software level, it is worth mentioning the development of a REST API compatible with different front-end, so that the complexity associated with Semantic Web technologies remains hidden to them, working as a proxy between the resources exposed with representations in well-known standard formats and the SPARQL queries on the existing Linked Data in the triple store. The prototype made has been designed to be expandable in the future. It currently integrates the trees of the IFN in the province of Valladolid, although it is expected to extend these data in the future with those of other provinces and even countries, as well as offering the possibility of adding and consulting technical forestry data such as dendrometric measurements.

KEYWORDS

forestry, social semantic annotation, API, Semantic Web, Linked Open Data

Agradecimientos

Quiero expresar mi gratitud a todas aquellas personas que me han ayudado a la realización de este Trabajo Fin de Máster y a lo largo de toda mi estancia en la Universidad. En primer lugar, al grupo GSIC/EMIC por ofrecerme la posibilidad de realizar este Trabajo Fin de Máster, a mis tutores y a José Miguel por los consejos y el tiempo dedicado.

Por útlimo, dar las gracias a las personas que están día a día a mi lado, a mis amigos y mi familia. Su apoyo y comprensión en los momentos difíciles ha sido fundamental para hacer posible este momento. Gracias.

Índice general

Índice de figuras

Índice de tablas

Capítulo **1**

Introducción

La Web Social (*Social Web*) [1] es una evolución de la Web donde los componentes centrales son las personas y los contenidos que crean. Abarca los servicios de redes sociales, por ejemplo Facebook, así como sitios de contenido compartido como Flickr. Su objetivo es conectar personas que comparten los mismos intereses, los cuales pueden ser muy variados, desde el ámbito de las relaciones de amistad hasta el empresarial, pasando por el científico y técnico, entre otros. Debido al número de posibilidades que ofrece, su popularidad se ha visto incrementada en los últimos años y, a medida que se han creado más sitios web sociales, el contenido ha aumentado y continúa haciéndolo actualmente, convirtiéndose en uno de los factores clave en el crecimiento exponencial de los datos existentes en la Web, así como un importante contribuidor a la variedad de tipos de datos existentes [2]. Las comunidades de usuarios generan contenidos, en contraposición a un modelo más centralizado y con grandes corporaciones como protagonistas, propiciando un cambio de tendencia en el rol de los usuarios de la web, de consumidores pasivos de contenido a participar en la creación de contenido, uniéndose además al papel de la ciencia ciudadana[1] cada vez más relevante en el panorama actual.

Típicamente, para que los datos puedan estar conectados y sea más fácil navegar, descubrir contenido y clasificarlo, el etiquetado o anotación es una práctica común en la Web Social [1]. Sin embargo, estas anotaciones tienen el inconveniente de ser difícilmente reutilizables porque son heterogéneas y no siguen formatos comunes. Esto provoca la fragmentación de comunidades y la creación de silos de datos individuales, dando lugar a sitios aislados, que no pueden interoperar entre sí. La principal razón de esta falta de interoperación es que para la mayoría de las aplicaciones, comunidades y dominios de la Web Social todavía no existen normas comunes para el conocimiento y el intercambio de información o la interoperación. Además, no permiten búsquedas específicas, debido a que, para un ordenador, estas anotaciones carecen de significado [1, cap. 4](se refiere a que un ordenador solo tiene en cuenta las páginas web existentes y los enlaces –hipervínculos– entre ellas, mientras que un humano interpreta la información contenida como conceptos y propiedades del mundo real). Esto supone una brecha de conocimiento que se puede reducir, en cierto modo, con la introducción de mecanismos de representación de conocimiento.

La Web Semántica [4] se propone para abordar los problemas anteriores y establece mecanismos de representación de conocimiento. Es una extensión de la *World Wide Web* (WWW) que difiere de la consideración tradicional de un sitio web como una colección de páginas, considerándolo como una colección de datos. Permite describir entidades, tales como personas y objetos, y las relaciones entre ellas, independientemente de su representación, de forma que sean procesables por un ordenador [1]. Tiene un fuerte potencial ya que es una plataforma muy útil para crear, reutilizar, operar y enlazar contenido dentro de un mismo sitio web social, así como con otros sitios web, que pueden ser, además, heterogéneos. Respecto a las anotaciones, si estas incluyen semántica, es decir, se combinan con esquemas de modelado

[1] La ciencia ciudadana es el área de investigación científica que incluye participación activa de público no especializado en sus proyectos [3].

formal como ontologías, se tienen anotaciones semánticas. La particularidad es que estas anotaciones no solo tendrán significado para los humanos, sino también tendrán información legible para las máquinas [1, 5].

La combinación de la Web Social y la Web Semántica tiene como resultado la Web Semántica Social, donde se promueven ciertas cualidades como la colaboración entre usuarios. Por ejemplo, esta colaboración se puede observar en los datos, ya que la mayoría de ellos, relacionados con la Web Social y la ciencia ciudadana, se engloban dentro de la categoría de datos abiertos (*Open Data*) [6], es decir, están disponibles públicamente para cualquier individuo u organización que desee consultarlos. Sin embargo, los datos abiertos son muchas veces difíciles de utilizar por no estar bien estructurados, por ser inconsistentes, o mal documentados. Mediante el uso de ontologías y otras tecnologías de la Web Semántica se proponen los Datos Abiertos Enlazados (*Linked Open Data*, LOD), una web distribuida a nivel de datos donde estos se enlazan con otros mediante identificadores globales y únicos [1, 5, 7].

1.1 Motivación

La expansión de la Web como un medio de intercambio de datos es incesable, motivada por el aumento de datos y su compartición en línea en todos los ámbitos, ya sean datos abiertos por parte de gobiernos de todo el mundo, científicos o procedentes de redes sociales. Desde el punto de vista del publicador, su objetivo es que los datos se puedan compartir de forma abierta o con acceso controlado, mientras que los consumidores de los datos –que a su vez pueden ser productores– necesitan encontrarlos, utilizarlos y vincularlos adecuadamente. En ocasiones, los esfuerzos realizados por los publicadores de los datos no satisfacen las necesidades de los consumidores, generando una situación problemática.

En el caso de la ciencia e ingeniería forestal, así como la silvicultura, la información con la que se trabaja son conjuntos de datos a gran escala que abarcan largos periodos de tiempo, debido a que los árboles tienen grandes longevidades y, para poder realizar estudios sólidos, se requiere información precisa resultante de un seguimiento continuo a lo largo de sus vidas [8]. Los datos son necesarios para las tareas de gestión y planificación forestal. Para recoger estos datos de forma oficial, los expertos, tales como ingenieros forestales, hacen uso de inventarios forestales gestionados por las distintas administraciones públicas. Los inventarios forestales continuos (como el IFN, Inventario Forestal Nacional, español) son los más valiosos porque permiten trazar el crecimiento de los árboles a lo largo de distintas ediciones de un inventario. Es necesario señalar que estos datos, a pesar de ser, en general, muy precisos (dependiendo de la metodología del inventario), habitualmente están aislados y descritos con esquemas de datos dispares o formatos poco comunes (debido al uso de sistemas legados y decisiones tecnológicas cuestionables) [8]. Por tanto, su consulta no es una trivialidad ya que requiere conocimiento del dominio y conocimientos técnicos relativos a modelos, formatos, bases de datos, etc. Por tanto, solo está al alcance de personas con los suficientes conocimientos para ello, excluyendo a otros tipos de interesados en estas materias como estudiantes o aficionados. Además, estos datos se actualizan cada cierto periodo de tiempo y solo recogen áreas específicas, resultantes de muestreos, elegidas por el interés que puedan suscitar en cuanto a control de incendios, plagas, etc. en compromiso con el coste que supone realizar este estudio detallado del ecosistema (realizar un inventario de todos los árboles es inviable económicamente). Por ejemplo, en el caso del IFN del Gobierno de España el ciclo es decenal y almacena mediciones elaboradas en los montes de todo el territorio nacional [9].

En relación a las limitaciones referentes a la conexión de los datos, los formatos y la complejidad de consulta, las tecnologías de datos abiertos enlazados y de la Web Semántica facilitan la integración de datos (por ejemplo, entre inventarios de diferentes ediciones, permiten conectar datos de inventarios con otros de cobertura arbórea o de especies...) y la accesibilidad de los datos forestales, permitiendo

consultas más complejas a causa de la semántica subyacente, es decir, el uso de ontologías comunes simplifica estas consultas al poder utilizar términos sin ambigüedad [1, 8].

Respecto a la condición de los datos restringidos a cierto tipo de territorios o parcelas y los periodos de actualización, las plataformas colaborativas y sociales, tales como las *wikis*, han demostrado el gran valor que tiene la producción de contenido de calidad a través de la comunidad, por lo que resulta atractiva la idea de sinergia entre instituciones y ciudadanos. Un espacio social es cualquiera donde se cree, anote y comparta contenido entre una comunidad de usuarios, por lo que, aplicando estos principios, aparte de consultar datos gubernamentales forestales, gracias a la colaboración ciudadana se podrían ampliar los datos inventariados y añadir imágenes de los árboles (característica novedosa, ya que el IFN no dispone de esta información ni otros inventarios). Este enriquecimiento de los conjuntos de datos forestales disponibles puede ser de interés en el marco de gestión administrativa, como por ejemplo a nivel de ayuntamientos contribuyendo a una gestión más eficiente de sus parques y jardines así como en el ámbito de la educación relacionada con temática forestal, donde se podrían plantear diversas tareas educativas como la identificación especies de árboles en los espacios verdes cercanos, con el objetivo de fomentar la participación del alumnado en la creación de contenido colaborativo y las discusiones al respecto.

La elaboración de un sistema enmarcado en la Web Semántica Social es ideal para atacar la casuística presentada, ya que reúne las condiciones necesarias para cumplir los requisitos deseados: facilidad de integración de datos, flexibilidad, heterogeneidad o variabilidad (datos de distintos tipos y fuentes: datos institucionales y de usuarios) y colaboración [1, 8].

1.2 Objetivos

El objetivo perseguido con la elaboración de este trabajo es el **diseño e implementación del servidor de una aplicación web socio-semántica de anotación de árboles** con un doble enfoque. En primer lugar, permitir la consulta de forma sencilla de datos forestales institucionales como los ofrecidos por el IFN [9] mediante la integración con la aplicación "Explorador Forestal" [10], una aplicación de consulta de datos oficiales del IFN pero que no permite creación de contenido. Adicionalmente, consultar datos y anotaciones creadas por usuarios (no institucionales). Todos estos datos serán datos abiertos enlazados y, en concreto, se pretende proporcionar información de árboles del territorio nacional español, aunque sería extensible a cualquier parte del mundo. En segundo lugar, registrar nuevos árboles no inventariados, anotar árboles existentes y/o sugerir actualizaciones a los datos oficiales mantenidos por instituciones gubernamentales. Los datos que se consideran de interés para este Trabajo Fin de Máster, en una primera aproximación, serían la posición de los árboles y su especie, además de imágenes de estos, construyendo una galería multimedia, basada en fotografías anotadas a partir de sus metadatos más la información adicional que incluya el creador.

Se pretende que sea una aplicación social, con participación abierta a todo tipo de usuarios de la comunidad forestal y ciudadanos en general, que sirva de herramienta de consulta, aprendizaje y colaboración, de forma que pueda convertirse en un nuevo sistema que contribuya a la expansión de la ciencia ciudadana y al mantenimiento de la biodiversidad, satisfaciendo ciertas necesidades o limitaciones existentes actualmente. Para poder cumplir con éxito los requisitos expuestos, para la parte más técnica del sistema, se emplearán tecnologías de la Web Semántica. Uno de los motivos de esta elección es la conversión de los datos del IFN3 a LOD que se ha realizado en el grupo de investigación GSIC [11]. Esto supone que la integración de este repositorio de datos del IFN con la aplicación que se plantea es inmediata mediante las tecnologías de la Web Semántica.

En este orden de ideas, las anotaciones que se creen sobre los árboles se anotarán con términos de una ontología de árboles permitiendo la integración de esta información de origen social con datos

oficiales. Además, las anotaciones serán interpretables por los ordenadores al utilizar mecanismos de representación de conocimiento y las consultas de los datos serán más fáciles debido al uso de los términos de la ontología sin ambigüedad. Estas anotaciones realizadas serán semánticas, con las ventajas en el etiquetado que esto supone, pero cuya complejidad estará oculta y será transparente para los usuarios finales que no tengan conocimientos de este tipo de tecnologías.

Este objetivo global puede verse desglosado en los siguientes subobjetivos:

- **Análisis y diseño del modelo de datos e integración con datos externos (IFN)**. Comprende el estudio del dominio, la elaboración de la ontología con los términos necesarios para el dominio de la aplicación y las tareas necesarias para integrar el repositorio de datos abiertos enlazados del IFN elaborado por el proyecto Cross-Forest [12].
- **Diseño e implementación del servidor de anotación socio-semántico**. Incluirá el diseño de la arquitectura del sistema a alto nivel y los recursos diseñados para la Interfaz de Programación de Aplicaciones (*Application Programming Interface*, API) que se va a exponer, así como su posterior configuración y codificación. Tendrá distintos componentes entre los que destaca la API, encargada de atender y recibir peticiones y ocultar las complejidades asociadas a la Web Semántica a posibles aplicaciones cliente que se integren con el servidor. Ofrecerá la posibilidad de consultar y crear datos de forma sencilla (métodos documentados) y con formatos estándares.
- **Evaluación funcional y de rendimiento**, a través de pruebas diseñadas para medir y evaluar el funcionamiento del servidor en conjunto.

1.3 Fases y Métodos

Desde un punto de vista general, para alcanzar los objetivos planteados, las labores a desarrollar se han ordenado siguiendo una aproximación sistemática de acuerdo con el método o diseño de la ingeniería [13], dando lugar a las siguientes fases:

I. Identificación del problema: comprende el reconocimiento de las limitaciones existentes en el ámbito forestal respecto a la consulta de datos inventariados de árboles y la inexistencia de iniciativas y aplicaciones sociales en este dominio.

II. Estudio de soluciones existentes, identificación de requisitos y restricciones:

a) Búsqueda y estudio de otras alternativas disponibles (aplicaciones forestales como "Explorador Forestal" [10], "Tree Talk" [14] y "Pl@ntNet" [15]), identificando las carencias que presentan en relación con el problema que se desea resolver.

b) Identificación y establecimiento de los requisitos y restricciones que se deben cumplir para satisfacer las necesidades de los usuarios.

III. Planteamiento de soluciones y definición de objetivos:

a) *Lluvia de ideas*, junto con los tutores, para desarrollar ideas en relación con la problemática identificada.

b) Definición de los objetivos perseguidos en este trabajo, así como de posibles líneas de trabajo futuras, y acotar el trabajo a materializar.

IV. Diseño de la solución planteada:

a) Diseño de un ontología específica para el dominio con términos propios y estudio y reutilización de otras ontologías bien conocidas y aceptadas para términos comunes.

b) Diseño de los datos expuestos e integración con datos del IFN.

c) Diseño de la arquitectura del sistema *software* y la API expuesta.

V. **Desarrollo, implementación y verificación:** incluye la instalación y configuración de los paquetes y componentes *software* requeridos para la puesta en funcionamiento del servidor o como condición necesaria para la lógica a generar o codificar. Posteriormente, tiene lugar la validación en conjunto de la aplicación resultante.

VI. **Resultados y lanzamiento**: compuestos principalmente por pruebas funcionales y de carga con objeto de demostrar el correcto funcionamiento del sistema o comportamiento esperado, además del lanzamiento de la aplicación restringido a miembros del laboratorio GSIC/EMIC [11].

En relación al modelo de datos, la metodología de creación de la ontología empleada se basa en [16] y comprende las siguientes fases:

I. **Dominio y alcance de la ontología:** se establece a partir del dominio, tras un estudio del contexto del proyecto y los datos que se requieren.

II. **Estudio de ontologías existentes para su reutilización:** a partir de los conceptos y relaciones identificados en el análisis del modelo de datos se realiza una etapa de búsqueda de ontologías existentes para su reutilización.

III. **Enumeración de términos importantes de la ontología:** los elementos que no se encuentren en ontologías existentes se modelan mediante una ontología propia, para ello se enumeran los principales conceptos y relaciones entre ellos.

IV. **Definición de clases y jerarquía de clases:** se establecen las clases a partir de los conceptos identificados, comenzando de las más generales a las especializaciones para poder generar una jerarquía.

V. **Definición de propiedades de las clases:** las propiedades se identifican a partir de las relaciones entre conceptos del dominio.

VI. **Definición de restricciones de propiedades:** el último paso en la creación de la ontología de este trabajo se corresponde con la definición de restricciones de dominio y rango sobre las propiedades que se han creado.

Respecto al desarrollo del *software* se han seguido las fases establecidas por los procesos de ingeniería de *software*, las metodologías ágiles y el desarrollo de *software* incremental [17, cap. 2, 3], donde los procesos de especificación, diseño e implementación se solapan. Las actividades fundamentales son:

I. **Especificación:** se define la funcionalidad del *software* y las restricciones en las operaciones. Comprende la etapa de análisis donde se da una visión general del contexto del proyecto, el sistema que se plantea como solución, sus requisitos, casos de uso y modelo de datos.

II. **Desarrollo:** integra los procesos relativos a cómo el *software* satisface los requisitos establecidos y se produce, es decir, el diseño e implementación del sistema.

III. **Validación:** procesos que sirven para comprobar y asegurar que el *software* satisface las necesidades iniciales. Abarca la fase de pruebas y evaluación.

Fundamentalmente ha sido un trabajo iterativo, teniendo en cuenta la retroalimentación recibida por los tutores de este Trabajo Fin de Máster y estableciendo prioridades a las tareas y actividades en función del horizonte temporal. Adicionalmente, hay que señalar la existencia de una importante **etapa de formación** extendida a lo largo de todo el desarrollo de este Trabajo Fin de Máster, la cual comprende el estudio de los conceptos básicos y, en otros casos, avanzados en varias áreas necesarias para la elaboración de este trabajo. Principalmente, ha sido necesario un estudio de los fundamentos de la Web Semántica y sus tecnologías (RDF [18], RDFS [19], OWL[20] y SPARQL [21]), programación de aplicaciones web, profundizando en programación asíncrona y lenguaje de programación JavaScript, servidores web Nginx [22], fundamentos de contenedores Docker [23] y algunos conceptos de administración de sistemas Linux.

1.4 Medios

Para la realización de este proyecto ha sido necesario el acceso a las herramientas *hardware* y *software* que se detallan a continuación:

Hardware

- PC portátil con las siguientes características:
  - Procesador Intel® Core™ i7-3557U @ 2.00 GHz.
  - 12 GB de memoria RAM.
  - Disco SSD de 500 GB de capacidad.
- A partir del equipo anterior, se realiza una conexión remota al servidor establecido en una máquina virtual alojada en la nube de cómputo que se va a especificar en el apartado de *Software*, de forma directa a través de la red del laboratorio o a través del siguiente servidor del grupo GSIC/EMIC de la Universidad de Valladolid:
  - Titán, es la pasarela a la red privada del laboratorio y tiene las siguientes características:
    - Procesador Intel® Xeon Irwindale @ 3.20 GHz.
    - 8 GB de memoria RAM.

Software

- Infraestructura necesaria en la nube de cómputo Openstack del grupo GSIC/EMIC para la creación de una máquina virtual que funcionará como servidor [24]. Para permitir su conexión con el exterior es necesario crear y configurar como mínimo un *router* virtual y uno o varios grupos de seguridad de Openstack. Se ha dotado a esta instancia de una IP flotante perteneciente a la red privada del laboratorio y se le ha asignado el nombre *timber* dentro del dominio *gsic.uva.es* en el servidor DNS. Las características del servidor son:
  - 2 VCPUs (CPUs virtuales).
  - 4 GB de memoria RAM.
  - Disco duro de 40 GB montado como volumen.
- Visual Studio Code [25]: editor ligero de código fuente disponible para Windows, MacOS y Linux con soporte incorporado para Javascript, TypeScript y Node.js. Dispone de una tienda con múltiples extensiones para extender sus funciones. Se ha utilizado también como editor de LaTeX.
- Protégé 5.5.0 [26]: plataforma gratuita y de código abierto que proporciona un conjunto de herramientas para construir modelos de dominio y aplicaciones basadas en el conocimiento con ontologías.
- WebVOWL 1.1.7 [27]: aplicación web para la visualización interactiva de ontologías. Implementa la Notación Visual para Ontologías OWL (VOWL) proporcionando representaciones de gráficas para elementos del Lenguaje de Ontología Web (OWL).
- Ontology Visualization [28]: aplicación para representar triplas RDF gráficamente.
- Astah UML 8.2.0 [29]: herramienta de diseño *software* que soporta UML, diagramas ER, diagramas de flujo de datos, diagramas de flujo, mapas mentales y más.

- Docker versión 19.03.12 [23]: plataforma de *software* que permite crear, probar e implementar aplicaciones rápidamente, ya que empaqueta los componentes necesarios en la ejecución del *software* (bibliotecas, herramientas de sistema, código y tiempo de ejecución entre otros) en unidades estandarizadas denominadas contendedores.
- Imagen docker Nginx 1.17.10 [30]: contenedor con imagen de servidor web Nginx [22].
- OpenLink Virtuoso 7.2.5 Open-Source Edition [31]: una plataforma híbrida de almacén de triplas y *middleware* basada en RDBMS (*Relational Database Management System*) de alto rendimiento y escalable para el desarrollo de datos enlazados. Soporta SPARQL integrado en SQL para consultar datos RDF almacenados en la base de datos de Virtuoso.
- LaTeX[32]: sistema de composición de textos, orientado a la creación de documentos escritos.
- MATLAB R2017b [33]: sistema de cómputo numérico para la manipulación de vectores y matrices, así como la representación de datos y funciones.

Bibliografía

- Hemerotecas de la Escuela Técnica Superior de Ingenieros de Telecomunicación.
- Bases de datos electrónicas suscritas por la Universidad de Valladolid.
- Documentación y manuales en formato electrónico disponibles en Internet.

1.5 Estructura del documento

El documento se encuentra dividido en 7 capítulos, entre los que se incluye el presente capítulo. Su contenido se describe brevemente a continuación:

Capítulo 1 - Introducción. Se expone la temática del documento, así como la motivación, los objetivos perseguidos, las fases y medios necesarios para alcanzarlos y la estructura del presente documento.

Capítulo 2 - Conocimientos previos y estado del arte. Se aborda el uso de laWeb como medio de intercambio de datos y se plantean las ventajas y problemáticas asociadas a la publicación de datos en la Web. Se tratan las formas de publicar los datos y se explica la base teórica de los principios de diseño REST (*Representational State Transfer*) [34]. Posteriormente, se presentan los datos abiertos y LOD y la Web Social con sus principales problemas. Se centra además en las tecnologías de la Web Semántica, sus fundamentos y cómo se pueden aplicar, dando lugar a las aplicaciones sociosemánticas y a la anotación semántica. Estos conceptos se introducirán con objeto de proporcionar un contexto al lector y justificar su aplicación en este trabajo. Finalmente, se concreta el área de dominio del trabajo y se comentan los problemas asociados a los datos forestales, su publicación y repositorios conocidos. También se analizan varias aplicaciones existentes en el ámbito forestal como estado del arte.

Capítulo 3 - Análisis. Se presenta una visión global del proyecto a realizar y el sistema propuesto como solución, presentando someramente la aplicación completa para, posteriormente, centrarse en el sistema a analizar, el *back-end*. A lo largo del análisis se realizan las tareas de especificación de los requisitos, diseño y desarrollo de los casos de uso y análisis del modelo de datos (ontología).

Capítulo 4 - Diseño del back-end de anotación semántica de árboles. Este capítulo expone el diseño completo de los distintos componentes. Incluye la arquitectura global del sistema, detallada por componentes; el estudio de ontologías conocidas para el dominio de la aplicación con el objetivo de reutilizarlas; el diseño de la ontología con los términos específicos y las IRIs (Identificador Internacionalizado de Recursos, *Internationalized Resource Identifiers*) [35]. Posteriormente, presenta los datos que se expondrán y la integración de datos externos en el sistema y cómo se lleva a cabo.

Por último, se trata el diseño de la API REST, presentando su modelo de recursos, así como las IRIs e identificadores de estos.

Capítulo 5 - Implementación y despliegue del servidor. Contiene los detalles relativos a la implementación de la parte servidora y su despliegue. Para ello se seleccionan las tecnologías concretas a utilizar y se justifica estas decisiones. A continuación, se presentan los servicios implementados y/o configurados para poder ofrecer cierta funcionalidad, prestando especial atención a la API REST ofrecida, profundizando en cómo se ha estructurado el código y las principales librerías utilizadas en su desarrollo. Para terminar, se explica la puesta en producción del sistema y la aplicación de buenas prácticas recomendadas.

Capítulo 6 - Validación y evaluación. Se prueba el sistema desarrollado. En una primera sección, en cuanto a funcionalidad, es decir, se demostrará que el comportamiento es el esperado para distintos casos de prueba. En una segunda sección, a nivel de rendimiento, con la realización de distintas pruebas de carga automáticas para comprobar qué ocurre cuando el número de usuarios concurrentes crece, así como el número de peticiones que debe atender. A su vez, se realizará un estudio de la influencia de las cachés en el sistema ofrecido. Finalmente, se mostrará cómo podría integrarse con una aplicación cliente y un ejemplo concreto de integración con un *front-end* realizado por [36].

Capítulo 7 - Conclusiones y líneas futuras. Este capítulo final recoge las principales conclusiones extraídas con la elaboración de este Trabajo Fin de Máster y se proponen posibles líneas de trabajo futuro manifiestas a partir de su realización y las limitaciones actuales que presenta.

Además de un capítulo en el que se recogen las fuentes bibliográficas y referencias consultadas para la redacción de este documento, se añaden unos apéndices cuyo contenido se expone en las líneas siguientes:

Apéndice A - *Ontología STA (Social Tree Application)*. Se presenta la ontología desarrollada con los términos específicos de anotación social de árboles.

Apéndice B - *Documentación API REST*. Incluye la documentación detallada de la API REST diseñada e implementada en este Trabajo Fin de Máster.

Apéndice C - *Guía para OpenLink Virtuoso OpenSource Edition 7.2.5*. Contiene los detalles de la instalación y configuración del almacén de triplas y punto SPARQL.

Apéndice D - *Ficheros unit de configuración de servicios de systemd*. Indica los servicios *systemd* creados en el servidor para su funcionamiento en producción y sus ficheros de configuración.

Apéndice E - *Configuración de Nginx*. Se muestra el fichero de configuración empleado para la definición de los *host* virtuales y las opciones del servidor Nginx configurado como servidor web y *proxy* inverso.

Apéndice F - *Boxplots de tiempos de respuesta del sistema*. Muestra figuras con *boxplots* elaborados a partir de diversos experimentos de pruebas de carga realizados sobre el sistema con cargas sintéticas para distinto número de usuarios concurrentes. Se evalúan los tiempos de respuesta para diferente tipo de peticiones,tanto de creación como de lectura.

Capítulo 2

Conocimientos previos y estado del arte

Este capítulo se divide en tres partes, la primera de ellas aborda el uso de la Web como medio de intercambio de datos, las dificultades y desafíos a los que se enfrentan los publicadores y consumidores de datos, métodos de acceso, fundamentos de datos abiertos y datos enlazados, la Web Social y el etiquetado y sus principales problemas. Por otro lado, se desarrolla el concepto de Web Semántica, sus fundamentos, tecnologías y la anotación semántica. Finalmente, en la última sección se tratan temas relacionados con datos forestales, cómo se publican, algunos repositorios conocidos y aplicaciones que utilizan estos datos forestales para inventariado, etiquetado de árboles o con fines sociales.

2.1 Publicación de datos en la Web

La Web [37] es un espacio de información en el que los elementos de interés, denominados recursos, se determinan mediante identificadores globales llamados Identificadores Uniformes de Recursos (URI, *Uniform Resource Identifier*) [38]. El protocolo de transmisión estándar de la Web es es el Protocolo de Transferencia de Hipertexto (HTTP, *Hypertext Transfer Protocol*) [39] y se emplea para transferir solicitudes de hipertexto e información entre los servidores y los clientes de la Web (como los navegadores). Estos tres elementos constituyen la base arquitectónica de la Web y son características fundamentales de los principios de diseño REST de las APIs web (ver apartado 2.1.1.1).

Uno de los medios de intercambio de datos que se encuentra en crecimiento en la Web, ya que cada vez se publican más datos en ella. Este incremento de datos en línea está motivado por distintas fuentes y algunos ejemplos son: el aumento de datos abiertos compartidos por gobiernos de todo el mundo, el incremento de publicaciones y datos de investigación *online*, el análisis y publicación en línea de los datos de medios o redes sociales y la creciente presencia en la Web de importantes colecciones de patrimonio cultural, entre otros [2].

En términos generales existen dos roles principalmente en torno a la publicación de datos en la Web [2]: los publicadores o editores de datos y los consumidores. En ocasiones estos roles pueden solaparse, es decir, una misma persona u organización podría ser a la vez publicador y consumidor de datos. Las necesidades de ambos son bien distintas, pero deben converger para que se produzca un entendimiento entre ellos. Mientras que los publicadores quieren compartir datos, de forma abierta o con acceso controlado; los consumidores lo que desean es ser capaces de encontrar los datos que buscan, poder utilizarlos y vincularlos o enlazarlos. Además, anhelan que estos datos sean precisos, estén disponibles siempre y se actualicen de forma regular.

La amplitud y flexibilidad de la Web crea desafíos que afectan tanto a los publicadores como a los consumidores de datos y están relacionados con cómo representar, describir y poner a disposición los

datos de manera que su búsqueda y comprensión sea sencilla. Estas dificultades se sustentan en que la Web, para con los datos, permite la existencia de múltiples formas de representarlos y acceder a ellos. Por ello, las buenas prácticas recomendadas por organizaciones expertas como la W3C [2] en aspectos relacionados con la publicación y el consumo de datos, como los formatos de datos, el acceso a los datos, los identificadores de datos y los metadatos son de vital importancia, ya que proporcionan mejoras en la compresión, capacidad de procesar, capacidad de descubrir, reutilización, confianza, vinculación, acceso e interoperabilidad de los datos en la Web.

En general, la publicación de datos en la Web se resume en la Figura 2.1 [2]. Se tiene un conjunto de datos (*dataset*) que puede contener –y sería recomendable que incluyera– metadatos. Los metadatos pueden considerarse como "datos sobre datos" y son datos descriptivos estructurados sobre los recursos de la Web. Su uso es necesario ya que la hetereogeneidad de los datos publicados hace necesario proporcionar cierta información sobre los conjuntos de datos y las distribuciones y, también, pueden contribuir a la fiabilidad y la reutilización. Estos datos se publican en diferentes distribuciones, siendo una distribución la forma física específica de un conjunto de datos para facilitar el intercambio de datos a gran escala y satisfacer a distintos consumidores. Un aspecto importante de la publicación y el intercambio de datos en la Web se refiere a la base arquitectónica de la Web [37]. Tiene especial relevancia el principio de identificación que dice que los URIs [38] deben ser usados para identificar recursos. Por último, para promover la interoperabilidad entre los conjuntos de datos es importante adoptar vocabularios y normas de datos.

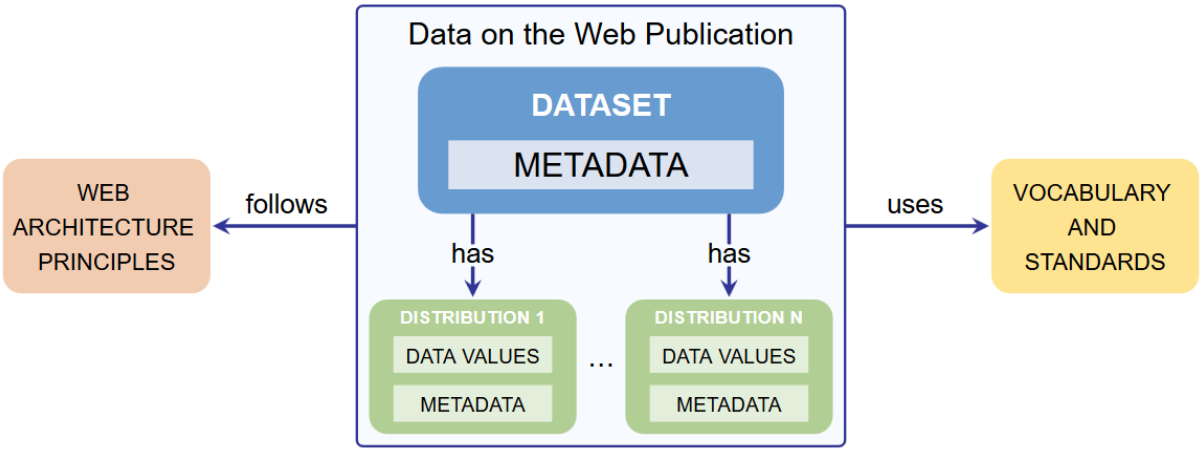


Figura 2.1: Publicación de datos en la Web [2].

2.1.1 Desafíos de los datos en la Web

Los principales desafíos respecto a la publicación de datos en la Web están relacionados con: procedencia y calidad, versionado, identificación, formatos, vocabularios, acceso, preservación, retroalimentación, enriquecimiento y republicación. A continuación se van a describir brevemente cada uno de ellos [2]:

- **Metadatos:** son esenciales ya que proporcionan información adicional que ayuda a los consumidores de datos a comprender mejor el significado de estos, su estructura y a aclarar otras cuestiones, como los derechos y las condiciones de la licencia, la organización que los generó, la calidad de estos, los métodos de acceso y el calendario de actualización de los conjuntos de datos. En ausencia de un contexto específico los datos carecen de significado y no podrán ser descubiertos ni reutili-

zados a excepción del editor. Además de diferentes niveles de granularidad, los metadatos pueden ser de varios tipos y legibles por humanos o interpretables computacionalmente (por máquinas) en función del formato. Si se definen metadatos interpretables por máquinas es recomendable reutilizar términos estándar y los vocabularios populares existentes. Por ejemplo, los términos de los metadatos de Dublin Core (DCMI) [40] y el vocabulario del catálogo de datos [41] pueden utilizarse para proporcionar metadatos descriptivos.

- **Licencias:** elegidas por los publicadores, permiten establecer restricciones en el uso y compartición de los datos y pueden ser parte de los metadatos o estar recogidas en un documento específico.
- **Procedencia y calidad:** El productor de los datos puede no ser necesariamente su editor, por lo que la recopilación y transmisión de metadatos correspondientes a la procedencia es particularmente importante. Sin procedencia, los consumidores no tienen ninguna manera inherente de confiar en la integridad y credibilidad de los datos que se comparten. Los editores de datos, a su vez, deben ser conscientes de las necesidades de las posibles comunidades de consumidores para saber qué grado de detalle es apropiado. Del mismo modo, la inclusión de información sobre la calidad de los datos es primordial, ya que tendrá impacto en las aplicaciones que los utilizan.
- **Versionado:** está relacionado con el hecho de que los datos publicados en la Web pueden cambiar con el tiempo. Para hacer frente a estos cambios, se pueden crear nuevas versiones de un conjunto de datos; aunque no hay consenso sobre cuándo los cambios en un conjunto de datos deben hacer que se considere un conjunto de datos totalmente diferente en lugar de una nueva versión.
- **Identificación:** la identificación y el uso de URIs persistentes en los conjuntos de datos es fundamental. La adopción de un sistema de identificación común permite que cualquier interesado pueda identificar y comparar los datos básicos de manera fiable. Es una condición previa esencial para la gestión y la reutilización adecuada de los datos. Además, el valor de los datos se incrementará si pueden vincularse con otros. Para ello es necesario que estos puedan ser referidos por otros conjuntos de datos. Cuando esos identificadores son URIs HTTP [38], pueden ser desreferenciados y se pueden descubrir más datos. Esta idea es uno de los fundamentos de los datos enlazados (*Linked Data*) [42], donde un dato se enlaza con otro, y de *Hypermedia* (ver 2.1.1.1), donde los enlaces pueden ser a más datos o a servicios que pueden actuar o relacionarse con los datos de alguna manera, dando lugar a la Web de Datos. Estos conceptos se desarrollan en detalle en la sección 2.1.3.
- **Formatos:** el formato en el que los datos se ponen a disposición de los consumidores es un aspecto clave para hacer que esos datos se puedan utilizar. El W3C fomenta el uso de formatos que puedan ser utilizados por la mayor audiencia posible y procesados fácilmente por los sistemas informáticos, ya que, a medida que los datos se hacen omnipresentes y los conjuntos de datos se hacen más grandes y complejos, el procesamiento por computadoras se hace cada vez más crucial. El uso de formatos de datos no estándar es costoso e ineficiente y los datos pueden perder significado a medida que se transforman. En cambio, los formatos de datos normalizados permiten la interoperabilidad, así como usos *a posteriori* como la visualización.
- **Vocabularios:** definen los conceptos y relaciones utilizados para describir y representar un área de interés. Se utilizan para clasificar los términos que pueden utilizarse en una aplicación particular, caracterizar las posibles relaciones y definir las posibles limitaciones en el uso de esos términos. Se han acuñado varios cuasisinónimos de "vocabulario", por ejemplo, ontología, vocabulario controlado, tesauro, taxonomía, lista de códigos, red semántica... Aunque no existe una división estricta de estos conceptos, ontología tiende a denotar los vocabularios de las clases y propiedades que estructuran las descripciones de los recursos en los conjuntos de datos (enlazados). Las ontologías son los bloques de construcción clave de la Web Semántica (ver sección 2.2).
- **Acceso:** proporcionar un acceso sencillo a los datos en la Web permite tanto a los humanos como a las máquinas aprovechar los beneficios de compartir los datos utilizando la infraestructura de la

Web. Por defecto, la Web ofrece acceso mediante métodos de HTTP [39]. Los mecanismos existentes son muy variados en cuanto a complejidad desde la simple descarga masiva de un archivo o, cuando los datos se distribuyen en varios archivos o requieren métodos de recuperación más sofisticados, mediante una API. En el apartado 2.1.1.1 se profundiza en el concepto de API.

- **Preservación:** tiene que ver con la disponibilidad y almacenamiento de los datos en la Web en todo momento en un futuro. No es realista suponer que todos los datos estarán permanentemente disponibles para siempre, sin embargo, los editores pueden indicar que estos datos han sido eliminados o archivados para informar al consumidor.
- **Retroalimentación:** es la información que los consumidores de datos proporcionan los editores en relación sus experiencias de uso, preferencias y necesidades. Esto ayuda a los editores de datos a mejorar la integridad de sus datos publicados, además de fomentar la publicación de nuevos datos y les permite conocer si los datos publicados satisfacen las necesidades de los distintos consumidores basándose en la opinión de los usuarios. Por otro lado, proporciona a los consumidores un mecanismo de poder expresar sus opiniones.
- **Enriquecimiento y re-publicación:** se refiere a un conjunto de procesos que pueden utilizarse para aumentar, perfeccionar o mejorar de otro modo los datos en bruto o previamente procesados. Se pueden rellenar los valores que faltan y añadir nuevos atributos y medidas a partir de los datos brutos existentes. Los conjuntos de datos también pueden enriquecerse reuniendo resultados adicionales de la misma manera que los datos originales, o combinando los datos originales con otros conjuntos de datos. Esto ahorra tiempo a los usuarios y fomenta más tipos de reutilización, debido a que el conjunto de datos es más valioso. También se incluye la reutilización de datos como otra forma de publicar los datos o combinarlos con otros conjuntos de datos para crear aplicaciones o visualización en la Web o como una traducción. En el capítulo 4 se detallará cómo se ha realizado la reutilización de datos provenientes del IFN y enriquecidos por los desarrolladores del "Explorador Forestal" [10] en la aplicación de este Trabajo Fin de Máster.

2.1.1.1 API como método de acceso a los datos

Una API es una abstracción implementada en el *software* que define cómo otros deben hacer uso de un paquete de software como una biblioteca u otro programa reutilizable. Las APIs se utilizan para proporcionar a los desarrolladores acceso a los datos y la funcionalidad de un sistema determinado [6].

En relación con el acceso a los datos, uno de los medios más versátiles son las APIs. Se utilizan en caso de datos generados en tiempo real (o casi real, en sentido menos estricto), de forma que proporcionan un sistema automatizado de acceso inmediato a los datos. Pero su uso no se limita a datos en tiempo real, ya que las APIs son adecuadas para todo tipo de datos en la Web. Aunque inicialmente la construcción de una API requiere un cierto trabajo, mayor que otras alternativas como la publicación de archivos para su descarga, los publicadores de datos están de acuerdo en que, en general, merece la pena ofrecer una API bien documentada, basada en normas y estable, ya que tiende a ser el enfoque más flexible para servir subconjuntos de datos [2]. Esta flexibilidad viene dada por las personalizaciones en los datos que se proporcionan en cada situación a los consumidores y la capacidad de procesamiento de estos. En los casos de conjuntos de datos de grandes volúmenes, actualizaciones frecuentes o gran complejidad, la API se presenta como la mejor opción para publicarlos [2]. De acuerdo con lo anterior, una API también será la mejor alternativa para la integración con otras aplicaciones web, como podría ser una interfaz web para la visualización de datos o interfaz gráfica de interacción con usuarios, ya que se podrán obtener datos específicos mediante la consulta sistemática a una interfaz bien definida. En concreto, esto se ajusta al caso que se aborda en este Trabajo Fin de Máster y se aplicará en el diseño del sistema (ver capítulo 4).

Uno de los estilos arquitectónicos más conocidos que se aplican a APIS web es REST [34], donde se emplean los mismos principios arquitectónicos de la Web para los diseños de estas. Cabe señalar que

no necesariamente una aplicación que respete los principios de REST tiene que utilizar tecnologías web, aunque en el presente caso así sea. La popularidad de REST está motivada por las ventajas que ofrece a las aplicaciones construidas: gran escalabilidad, buenas prestaciones en tiempo de ejecución, acoplamiento débil entre clientes y servidores y facilidad de modificación [43].

Las características de las aplicaciones basadas en REST son las siguientes [43]: arquitectura cliente-servidor; basadas en comunicación sin estado[1]; los datos y la funcionalidad se consideran recursos; los recursos se identifican mediante identificadores globales (si se utilizan los estándares de la Web son URIs [38]); presentan una interfaz uniforme, es decir, se actúa sobre los recursos mediante un conjunto de operaciones simples predefinidas (como los métodos del protocolo HTTP [39]: GET, PUT, POST, DELETE...); los recursos están desacoplados de su representación (contenido accesible en varios formatos) y pueden incluir metadatos (por ejemplo, en cabeceras HTTP, para negociar el formato de representación entre cliente y servidor); y, aunque las interacciones sean sin estado, se puede transferir de forma explícita mediante el uso de enlaces, siendo el cliente responsable de mantener su estado de interacción con el servidor y el servidor encargado de mantener el estado de los recursos. De esta forma se evita acoplamiento entre solicitudes y respuestas y se facilitan el mantenimiento y comprensión de la API por parte de los desarrolladores [2].

2.1.2 Datos abiertos

Como se ha explicado en la sección 2.1.1 el acceso a los datos no es fácil, ya sea por restricciones en el acceso o dificultades en el uso de estos, relacionadas con formatos, etc. Los datos abiertos persiguen el objetivo de que estos estén disponibles de forma generalizada, sin restricciones, para cualquier individuo u organización que desee consultarlos. Su origen se encuentra en lo que se conoce como "movimiento de datos abiertos" cuya importancia se ha disparado desde finales de la década de los 2000, debido a que se han adherido gobiernos e instituciones de todo el mundo con el objetivo de publicar sus datos a libre disposición, para fomentar las sociedades abiertas, mediante la rendición de cuentas y la transparencia y en formatos que permitan su reutilización [44].

Aunque los datos abiertos pueden provenir de cualquier fuente, los grandes contribuidores se encuentran en la ciencia, algunas empresas privadas y, fundamentalmente, las administraciones públicas. Este último ha dado lugar a lo que se denomina *Open Government Data* (OGD) [45], una filosofía y un conjunto de políticas que promueven la transparencia, la responsabilidad y la creación de valor poniendo los datos del gobierno a disposición de todos.

Los datos abiertos se pueden publicar en diversos medios como tablones de anuncios, páginas web y aplicaciones móviles, documentos, etc. Actualmente, la publicación de datos en la Web es el medio más factible por múltiples razones como el volumen de datos almacenados y la accesibilidad (entendida como facilidad de acceso). No obstante, la publicación de datos en la Web trae consigo una serie de desafíos y dificultades (ver sección 2.1.1), por lo que se requieren nuevas formas de exponer, compartir y conectar datos en la Web. En este contexto se proponen los datos abiertos enlazados (*Linked Open Data*, LOD) [44].

2.1.3 Datos enlazados y datos abiertos enlazados

Los datos enlazados (*Linked Data*, LD) se definen formalmente de acuerdo con la W3C [6] como *«Un patrón para hipervincular conjuntos de datos interpretables por máquina entre sí utilizando técnicas de la Web Semántica, especialmente a través de el uso de RDF (Resource Description Framework) y URIs.*

[1] Se refiere al estado de la sesión

Permite consultas SPARQL distribuidas de los conjuntos de datos y un enfoque de exploración o descubrimiento para encontrar la información (en comparación con una estrategia de búsqueda). Los datos enlazados están destinados a ser accesibles tanto para los humanos como para máquinas. [...] Utilizan la familia de estándares RDF para el intercambio de datos (por ejemplo, RDF/XML, RDFa, Turtle) y la consulta (SPARQL)». Básicamente, LD surge como una solución a un problema previo a la existencia de normas internacionales de intercambio de datos en la Web: la consulta de datos de múltiples fuentes y su combinación sin un esquema único común compartido por todos los datos (habrá múltiples esquemas y algunos competidores entre sí, ya que pertenecen al mismo dominio). Proporciona un mecanismo sencillo para combinar datos de múltiples fuentes en la Web y tiene como principal objetivo establecer normas internacionales de la Web para la publicación, difusión y reutilización de datos estructurados.

Los cuatro principios propuestos por Tim Berners-Lee en los que se basan los datos enlazados son los siguientes [42]:

1. Identificar los recursos con URIs.
2. Usar URIs HTTP para que las personas y agentes puedan localizar y consultar estos recursos (desreferenciar).
3. Cuando alguien busca un URI, proporcionar información útil acerca del recurso cuando el URI ha sido desreferenciado, utilizando los estándares web abiertos como RDF y SPARQL.
4. Incluir enlaces a otros recursos relacionados usando sus URIs cuando se publiquen en la Web, para poder conectar los datos de una web con otra.

Otro término relacionado es datos abiertos enlazados [42, 6]. Es una particularidad del primero, son datos enlazados publicados en la Web pública bajo licencias abiertas. De acuerdo con lo anterior, LOD convierte a la WWW en una base de datos mundial, a veces denominada Web de Datos (*Web of Data*). La Web de Datos es, por lo tanto, un subconjunto de la WWW que contiene datos interpretables por máquinas representados como datos enlazados. Se profundizará en este concepto en la sección 2.2.

Bajo estos principios y con el objetivo perseguido que los datos enlazados sean de calidad, es decir, que cumplan estos requisitos propuestos, Tim Berners-Lee [42] sugiere un sistema de clasificación de estrellas (de una a cinco estrellas) para LOD[2]. Bajo este sistema se intenta cuantificar cómo de bien se ajustan los datos publicados en la Web a los principios de LD y sus buenas prácticas.

2.1.4 Web Social

La Web Social [1] se define como una plataforma de intercambio social y de colaboración con contribuciones reutilizables de la comunidad, en la que cualquiera puede publicar en masa utilizando *software* social basado en la Web y otros pueden suscribirse a la información, las noticias, los flujos de datos u otros servicios deseados. Su objetivo es conectar personas que comparten los mismos intereses, los cuales pueden ser muy variados, desde el ámbito de las relaciones de amistad hasta el empresarial, pasando por el científico y técnico, entre otros. Ha captado el interés de millones de usuarios, así como de empresas que han invertido millones de dólares en su adquisición [1]. En consecuencia, este auge ha convertido a la Web Social en una de las fuentes de generación masiva de datos en la Web junto con los datos abiertos de los gobiernos. Un punto de confluencia de la Web Social y los datos abiertos gubernamentales se encuentra en la ciencia ciudadana [3], que abarca proyectos diseñados por los científicos en los que los ciudadanos participan en la recogida de datos (proyectos contributivos), proyectos estructurados por los científicos en los que se ofrece a los ciudadanos oportunidades y herramientas para participar en el diseño del proyecto, la recogida de datos y su análisis (proyectos de colaboración); y proyectos en los que los

[2]Aunque se puede obtener un LD de cinco estrellas sin que sean datos abiertos, es obligatorio que sean datos abiertos para obtener cualquier estrella si se dice ser LOD.

ciudadanos participan en todas las etapas del proceso científico (proyectos cocreativos) [3].

El uso de estas tecnologías ha producido un cambio de tendencia en el rol del usuario de la Web respecto con los datos, de consumidor de contenido a participante activo en la creación de contenido (publicador). Algunos ejemplos populares son los artículos de Wikipedia, escritos y editados por voluntarios o Amazon, que utiliza la información sobre lo que los usuarios ven y compran para recomendar productos a otros usuarios. Además, el contenido compartido, anotado o discutido por lo usuarios y, por ende, los datos publicados en la Web, no se limita a contenido textual, sino también cualquier contenido multimedia como fotos, vídeos o incluso diapositivas de presentación. Por lo que se pone de manifiesto la heterogeneidad de los datos, cualidad a tener en cuenta a la hora de publicarlos en la Web [1].

En los sitios web sociales, la anotación o etiquetado de contenido [1] es una práctica común, ya que permite a los usuarios anotar fácilmente el contenido que publican o comparten con palabras clave de forma libre para que el contenido sea más fácil de navegar y de descubrir por otros, lo que lleva a un componente social de etiquetado. Su popularidad se debe a que es una forma ligera, ágil y evolutiva de anotar contenido. Una etiqueta (*tag*), normalmente, no es más que un descriptor de una sola palabra –una palabra clave asociada a un recurso en línea– y podría considerarse como un tipo de metadato generado por el usuario. Esto quiere decir que el usuario es quien voluntariamente añade las etiquetas y en ellas refleja sus necesidades o deseos, apuntando a lo que él mismo considera más importante en la información que ha compartido.

La principal ventaja del etiquetado social para los usuarios finales es que no necesitan aprender un esquema de vocabulario predefinido y cada uno puede usar las palabras clave que considere más oportunas. Además, la anotación se puede usar con varios propósitos desde la definición del tema, la expresión de una opinión hasta la propia autoreferencia (sea el caso que un usuario se etiqueta a si mismo en una foto de una red social) o el elemento social que un usuario desea enfatizar [1].

A pesar de todo, por sí solo, el etiquetado no suele ser suficiente, ya que a menudo se necesita más información sobre lo que alguien está hablando o se necesitan hacer preguntas más específicas sobre el tipo de contenido que se quiere. Las anotaciones tienen el inconveniente de ser difícilmente reutilizables y no permiten búsquedas específicas, debido a que, para un ordenador, carecen de significado (*a priori*, solo son legibles y comprensibles por humanos, no por máquinas). Los problemas típicos de las anotaciones no formales, es decir, libres son:

- Ambigüedad: es el caso en que una misma etiqueta, debido a que está descontextualizada, hace referencia a elementos distintos. Por ejemplo, buscar "*apple*" en una aplicación de fotos –sea Flickr– podría devolver imágenes de manzanas y también de dispositivos Apple, ya que hay coincidencia en la búsqueda de la palabra.
- Heterogeneidad: cuando varias etiquetas se refieren al mismo concepto u objeto son necesarias varias consultas para obtener todo el contenido relacionado con él. Las causas vienen dadas por la naturaleza multilingüe de las etiquetas, por el uso de acrónimos, versiones cortas o variaciones morfosintácticas (como sinónimos, plurales...). Un ejemplo de variaciones y heterogeneidad serían las etiquetas: *semanticweb*, *websemantique*, *sw*, *semweb*; todas ellas hacen referencia al mismo concepto, web semántica, pero son distintas y no se podrían recuperar todos los resultados asociados en una consulta única.
- Falta de organización entre etiquetas: se refiere a la falta de relación entre etiquetas, lo cual hace difícil la búsqueda de información relacionada en el caso de no estar consultando la etiqueta correcta. Por ejemplo, en el caso de usuarios con distintos niveles de conocimiento sobre una materia, si un experto etiqueta o anota un libro quizá use etiquetas más específicas que un usuario novel, el cual usará etiquetas más genéricas. A la hora de buscar ese libro, será complicado encontrarlo, ya que depende quién lo haya anotado la etiqueta será una u otra.

Por tanto, aunque la Web Social es una fuente de datos valiosos en la Web, presenta importantes limitaciones relacionadas con la interoperabilidad, además de los problemas expuestos sobre las anotaciones. Conlleva a la fragmentación de comunidades y la creación de silos de datos individuales, dando lugar a sitios aislados, que no pueden interoperar entre sí debido la inexistencia de normas comunes para el conocimiento e intercambio de información, problemática que ya ha sido enunciada para los datos de la Web de la sección 2.1.

2.2 Web Semántica

La Web ha creado un espacio de información común global pero, entre otros problemas, ha amplificado la sobrecarga de información y conocimientos al provocar la creación de más y más documentos web con hipervínculos. Mientras que un lector interpreta estas páginas e hipervínculos como representativos de conceptos y propiedades del mundo real, un ordenador no puede. De ahí que exista una brecha de conocimiento entre lo que está en la web y la interpretación que se puede hacer cuando se compara con lo que un ordenador puede deducir. Para cerrar la brecha de conocimiento, surgieron mecanismos de representación del conocimiento en la web más allá del HTML. Dado que la Web tiene como base una serie de propiedades únicas (como la distribución, la diversidad y la heterogeneidad), estas propiedades sirven como requisitos para los mecanismos de representación del conocimiento. Dadas las limitaciones anteriores, la Web Semántica [4]–también llamada Web 3.0 o Web de Datos– fue concebida para ayudar a la evolución del conocimiento humano como un todo, mediante la transformación de Internet de un contenido no estructurado a un mundo estructurado y semántico, con la posibilidad de conectar datos de diferentes fuentes y de modo que los ordenadores puedan interpretar y manipular datos en un formato estándar para ayudar a los humanos a dar sentido a la información. Como resultado se propuso LD [42], un patrón de hipervinculación de conjuntos de datos procesables por máquinas entre sí mediante técnicas de Web Semántica.

El término Web Semántica se refiere a la visión del W3C de la Web de datos enlazados. Las tecnologías de la Web Semántica permiten a las personas crear almacenes de datos en la Web, construir vocabularios y escribir reglas para el manejo de los datos. Los datos enlazados se intercambian y se consultan mediante RDF, OWL y SPARQL. Formalmente, la W3C [6] la define como: *«Una evolución o parte de la WWW que consiste en datos procesables por máquinas en RDF y una capacidad para consultar esa información de manera estándar (por ejemplo, a través de SPARQL)»*. Consiste en una web distribuida a nivel de datos donde unos datos apuntan a otros a través de URIs. Tiene un modelo de datos único y coherente que es parte de la infraestructura Web y representa los datos web distribuidos mediante un modelo de datos llamado RDF, que es la base para publicar y enlazar los datos. Esta infraestructura de datos permite a cualquiera expresar o añadir algún dato sobre alguna entidad de forma que pueda ser combinada con la información de otras fuentes. Para ello tiene que lidiar con un alto grado de interoperabilidad y es necesario que distintos puntos de vista puedan coexistir. Si la Web Semántica se considera como una base de datos global, entonces es fácil entender por qué se necesitaría un lenguaje de consulta. El lenguaje de consultas estándar de la Web Semántica es SPARQL [7].

La principal diferencia con los sitios web tradicionales (Web de documentos[3]) es que estos consisten en una colección de páginas (orientado a documentos), mientras que en el caso de un sitio web semántico (Web Semántica), este está compuesto por una colección de datos a partir de los cuales se generan las presentaciones. En el primer caso el centro son las presentaciones, es decir, los documentos en formatos HTML que se presentan pero que son difíciles de procesar computacionalmente, y en el segundo los datos estructurados con ontologías. Esta diferencia se puede comprender mejor con un ejemplo de búsqueda

[3]La WWW tradicional en la que los recursos publicados eran casi siempre documentos en lugar de datos interpretables por máquinas.

semántica. Por ejemplo, para responder a la siguiente pregunta: *¿de qué pueblo es el jugador que marcó el último gol de la final de la copa del mundo de 2010?*, un humano necesita analizar varias webs para llegar a una respuesta con la Web clásica, ya que las búsquedas se basan en palabras clave asociadas a documentos (páginas web), pero basta una consulta SPARQL a la DBpedia [46] como la que se muestra a continuación para obtener la respuesta.

```
SELECT ?pueblo
WHERE
{
    dbr:2010_FIFA_World_Cup_statistics dbp:lastGoalPlayer ?jugador.
    ?jugador dbo:birthPlace ?pueblo .
}
```

El modelado semántico viene motivado por el caos presente en los datos de la Web, por ejemplo en los datos abiertos y/o sociales. Gracias al uso de modelos, la información que anteriormente era muy compleja de encontrar y/o utilizar puede ser compartida, dando lugar a la cooperación y colaboración entre entidades o individuos. Permite organizar la información para el uso de la comunidad, proporcionando un marco para la comunicación humana, un significado para explicar conclusiones y una estructura para gestionar distintos puntos de vista [7]. Los lenguajes de modelado son: RDF [18], RDFS (*Resource Description Framework Schema*) [19] y OWL (*Web Ontology Language*) [20].

2.2.1 RDF

RDF es un lenguaje diseñado para la expresión de información arbitraria sobre cualquier entidad. Es el pilar fundamental de la Web Semántica para la representación de información en la Web, es decir, proporciona la base para representar y enlazar sus datos y el resto de estándares se construyen sobre el concepto de datos distribuidos. Se basa en la idea de identificar recursos –en el contexto de RDF, un recurso puede ser cualquier elemento del mundo del que alguien pueda querer hablar y se describa por un grafo RDF– mediante IRIs (habitualmente HTTP, aunque no es imprescindible) o literales y describiendo estos recursos en términos de propiedades simples y valores de estas. Se representan entidades y las relaciones binarias entre ellas [7, 6, 18].

Una aclaración sobre la identificación de recursos en la Web Semántica es que en la actualidad se identifican por IRIs según sus especificaciones [18], pero en el pasado se utilizaban URIs. Este cambio viene motivado por el hecho de que las IRIs son una generalización de las URIs, que permiten un rango más amplio de caracteres Unicode [47]. El uso de IRIs extiende la riqueza del vocabulario empleado en la identificación de recursos, en tanto que ofrece la posibilidad de utilizar idiomas distintos del inglés.

En RDF, la especificación del modelo de datos RDF, tiene como bloque de construcción básico la tripla o declaración (*statement*) y consta de dos entidades y una relación binaria entre ellas. Una tripla tiene tres partes: sujeto, predicado y objeto. Un conjunto de triplas forman un grafo RDF y este puede representarse visualmente como un diagrama de nodos y arcos dirigidos. El sujeto es el elemento sobre el que se hace la declaración, el predicado se corresponde con la propiedad específica de la entidad y el objeto ese el destino de esa relación. El modelo de datos de RDF [18] entre entidades (también llamadas recursos), que hace referencia a cualquier cosa denotada por un IRI o un literal, y términos RDF, que son solo IRIs, literales y *blank nodes*. El sujeto de una declaración es un IRI o *blank node*, predicado es una IRI y el objeto puede ser cualquiera de los términos RDF. Un ejemplo de triplas se muestra en la Tabla 2.1. Las triplas de la Tabla 2.1 se representan gráficamente en la Figura 2.2.

Anteriormente se introdujo RDF como base para los datos distribuidos. Para combinar información de múltiples fuentes de datos [7], sean por ejemplo dos grafos, se forma un grafo con todas las triplas de

Sujeto	Predicado	Objeto
Tree	hasAnnotation	Annotation
Tree	createdBy	IFN
Annotation	createdBy	IFN
Annotation	latitude	45.98
Annotation	longitude	3.01

Tabla 2.1: Triplas de ejemplo.

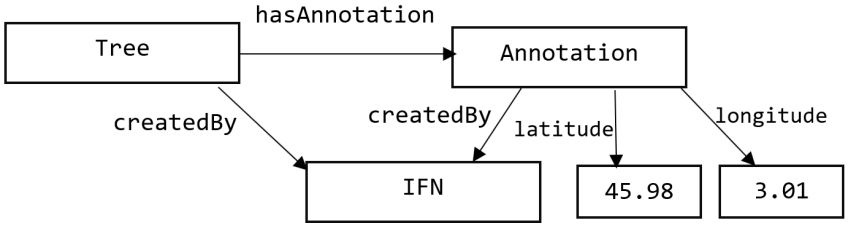


Figura 2.2: Ejemplo de visualización gráfica de triplas.

estos dos orígenes. Identificar cuando un nodo en un grafo es el mismo en otro es, en teoría, muy sencillo con RDF debido a que este problema ya ha sido resuelto con las tecnologías de la Web. Sin embargo, en la práctica suele ser problemático, aunque los IRIs sean identificadores globales para cada recurso, comunes a través de la Web, no siempre se mantienen, a veces son erróneos o se enlazan adecuadamente.

Para expresar IRIs más cortas por escrito se puede utilizar el esquema de IRIs abreviadas *qnames* [7]. Una URI expresada como un *qname* tiene dos partes: un espacio de nombres y un identificador, separados por un símbolo ":" entre medias. Por ejemplo, un *qname* sería: `sta:Tree` y hace referencia al IRI completo: `http://timber.gsic.uva.es/sta/ontology/Tree`. Este esquema tendrá especial utilidad a la hora de codificar ontologías, por ejemplo. Existen espacios de nombres estándar en la Web como `xsd:` o `xmlns:` y en la Web Semántica `rdf:`, `rdfs:` y `owl:`.

Por otro lado, como el intercambio de datos RDF representados en gráficamente no es adecuado, se necesita una sintaxis de serialización concreta para expresar RDF en forma textual, aunque el modelo de datos RDF no está ligado a ninguna en concreto. Las formas más conocidas son [7]: N-Triples, la más simple y directa a las triplas RDF en bruto; RDF/XML, en formato XML adecuada para infraestructura Web; y Turtle, la más compacta y adecuada para humanos y páginas impresas. En el presente documento se utilizará sintaxis Turtle cuando se requiera, por lo que se van a enunciar algunos fundamentos de sus sintaxis, dejando a un lado el resto de formas de serialización. En primer lugar, utiliza *qnames* y se suelen definir en el preámbulo. Las triplas se expresan como una lista de 3 recursos (sujeto, predicado, objeto) separados por un espacio y finalizado por un punto, un punto y coma o una coma (".", ";", ","); en función de si finaliza la tripla, la siguiente tripla tendrá el mismo sujeto o la siguiente tripla tendrá el mismo sujeto y predicado que la anterior, respectivamente. Otros detalles más específicos de RDF como los nodos en blanco (*blank nodes*) [18] quedan fuera del alcance de este documento.

2.2.2 Ontologías

Una ontología [6] se define como *«un modelo formal que permite representar el conocimiento para un dominio específico. Describe los tipos de cosas que existen (clases), las relaciones entre ellas (propiedades) y las formas lógicas en que esas clases y propiedades pueden utilizarse conjuntamente (axiomas)».* Básicamente estructuran datos enlazados (o abiertos enlazados) y se usan para modelar. Sirven para manejar puntos comunes y variabilidad. Se enmarcan dentro de los vocabularios utilizados para publicar datos en la Web. Si bien las palabras vocabulario y ontología suelen utilizarse indistintamente, la definición más estricta es que un vocabulario que contiene un conjunto de términos que se utilizan en un ámbito particular, que puede estructurarse (por ejemplo, jerárquicamente) como una taxonomía y combinarse con algunas relaciones, restricciones y reglas para formar una ontología. La combinación de una ontología junto con un conjunto de anotaciones constituye una base de conocimiento.

La aplicación o creación de ontologías consiste en definir todos los componentes de la ontología a través de un lenguaje de definición de ontología. Por lo general, se lleva a cabo en dos etapas: una etapa informal, en la que la ontología se esboza utilizando descripciones en lenguaje natural o alguna técnica de diagramas, y una etapa formal, en la que la ontología se codifica en un lenguaje formal de representación de conocimientos que es computable (comprensible y procesable) por una máquina (por ejemplo, RDFS u OWL). Para ello, se pueden utilizar distintas herramientas como Protégé [26]; sin embargo, la parte más importante es la documentación de la ontología para que sea comprensible por otros y se pueda reutilizar [1].

Los principales elementos de una ontología son [7]:

- Clases: representan conceptos y son conjuntos que contienen individuos. Las clases pueden organizarse en una jerarquía superclase-subclase (taxonomía).
- Individuos, entidades o instancias: representan objetos en el dominio y son miembros de una clase.
- Propiedades: relaciones binarias entre dos individuos o entre un individuo y un valor (literal).
- Literales: representan valores como cadenas, números o fechas.

Existen numerosas ontologías bien conocidas que pueden y deben reutilizarse como [1]: FOAF (*Friend of a Friend*) [48], para redes sociales; SKOS (*Simple Knowledge Organization System*); DC (*Dublin Core*) [40], para recursos en línea; o el Vocabulario Geográfico Básico del W3C4 para las coordenadas de las ubicaciones geográficas. También se pueden crear vocabularios personalizados para las necesidades propias como es el caso de este Trabajo Fin de Máster o adaptar una ontología publicada en la Web a necesidades propias y luego volver a publicar los cambios para que cualquiera pueda beneficiarse de ella. En la sección 4.2.2 se detallará la ontología creada.

2.2.3 Fundamentos de RDFS y OWL

RDFS [19] y OWL [20] son lenguajes de ontologías y difieren en sus niveles de expresividad, siendo OWL más expresivo que RDFS.

RDFS es un lenguaje que, a diferencia de RDF, modela conjuntos en vez de grafos y da nociones sobre cómo usar un grafo (no una estructura para representar datos): especifica el vocabulario, qué individuos están relacionados y cómo, las propiedades para definir relaciones entre conjuntos de individuos... RDFS extiende RDF introduciendo un conjunto de recursos distinguidos en el lenguaje (ciertas triplas tienen un significado especial y define el estándar), pero el lenguaje que utiliza es RDF y todo se expresa como triplas [7].

Los principales elementos de RDFS son las primitivas para describir las clases, idividuos y relaciones. Las clases se definen con `rdfs:Class` y las propiedades con `rdfs:Property`. Se pueden definir subclases y subpropiedades con `rdfs:subClassOf` y `rdfs:subPropertyOf`. Además se pueden restringir el dominio y el rango de una propiedad con `rdfs:domain` y `rdfs:range`. El dominio se utiliza para afirmar que cualquier recurso que tenga una propiedad determinada es un individuo de una o más clases, mientras que el rango se utiliza para afirmar que los valores de una propiedad son individuos de una o más clases. RDFS define otras primitivas de modelado: `rdfs:label`, permite definir una forma legible por el ser humano de un nombre; `rdfs:comment`, para comentarios; `rdfs:seeAlso` y `rdfs:isDefinedBy` se utilizan para indicar los orígenes de dirección relacionados. Por ejemplo, la propiedad `:hasAnnotation` del ejemplo anterior, se puede definir en RDF utilizando RDFS de la siguiente manera:

```
sta:hasAnnotation rdf:type rdf:Property ;
rdfs:label "has annotation"@en , "tiene anotación"@es ;
rdfs:comment "annotation of a tree"@en , "anotación de un árbol"@es ;
rdfs:domain sta:Tree ;
rdfs:range :Annotation ;
rdfs:isDefinedBy sta: .
```

Usando estas definiciones, los datos RDF pueden ser probados contra una especificación RDFS particular, de forma que al hacer una consulta se obtienen las triplas afirmadas y además las triplas inferidas a través del pequeño conjunto de inferencias definidas en el estándar de RDFS (como las relaciones de clase - subclase o propiedades - clases) [7, 19]. Las reglas de inferencia para RDFS son pocas y muy simples (por ejemplo se puede modelar mediante inferencias los conjuntos intersección y unión pero no de forma explícita). Por inferencia, en el contexto de la Web Semántica, se entiende que dada una información se puede determinar otra relacionada y considerar que ha sido establecida. De esta forma los datos estarán más integrados y serán más consistentes. En la Web Semántica, las propiedades se definen independientemente de las clases y las relaciones RDFS especifican que inferencias pueden hacerse sobre ellas en un contexto particular. El dominio y rango no se pueden usar para validar información solo para determinar nueva, basándose en la anterior.

En RDFS a diferencia de OWL no hay forma de afirmar que un individuo no es miembro de una clase –no hay noción de inferencia inconsistente [7]. Debido al conjunto limitado de capacidades de inferencia, RDFS es útil para combinar información de múltiples fuentes y orígenes pero no permite especificar cómo se tiene que combinar la información. Para poder llevarlo a cabo se utiliza OWL, que permite restricciones más elaboradas.

OWL –o la versión actual OWL 2 [20]– permite modelar axiomas avanzados dentro de las ontologías como la inversa de una propiedad (`owl:inverseOf`), indicar que una propiedad es simétrica (`owl:SymmetricProperty`) o transitiva (`owl:TransitiveProperty`). También se pueden definir clases o propiedades equivalentes con `owl:equivalentClass` y `owl:equivalentProperty`, respectivamente, y que dos individuos tienen la misma identidad (dos IRIs se refieren al mismo individuo) `owl:sameAs`. Las más importantes para la comprensión de es trabajo se exponen a través de ejemplos:

```
sta: a owl:Ontology ;
     owl:versionInfo 0.1 ;
     owl:imports ifn: .
```

Se utiliza para definir la IRI de la ontología (`owl:Ontology`) y su versión(`owl:versionInfo`). Se pueden importar otras ontologías para tener acceso a sus entidades, expresiones y axiomas (`owl:imports`).

```
sta:Tree a owl:Class .
```

Se pueden definir clases (`owl:Class`) y propiedades de tipo objeto para conectar pares de individuos (`owl:ObjectProperty`).

```
sta:hasAnnotation a owl:ObjectProperty .
```

Existen numerosos axiomas y restricciones adicionales en OWL que quedan fuera del alcance de este documento, ya que en este Trabajo Fin de Máster se va a utilizar principalmente RDFS con algunas extensiones de OWL para la creación de la ontología sección 4.2.2.

2.2.4 SPARQL lenguaje de consultas para RDF

El lenguaje de consultas estándar de la Web Semántica es SPARQL (Protocolo SPARQL y el Lenguaje de Consulta RDF) [21]. Es una familia de estándares del W3C y define un lenguaje de consulta para datos RDF, análogo al Lenguaje de Consulta Estructurado (SQL) para bases de datos relacionales y un protocolo de comunicación para consultas y resultados, por lo que el motor de consultas puede actuar como un servicio web [7]. Se caracteriza por ser un lenguaje de consulta de grafos, ya que los datos RDF se representan como tal. Además, proporciona extensibilidad dentro de los patrones de consulta (basado en el propio modelo de grafo RDF) y por lo tanto, capacidades de consulta avanzadas basadas en esta representación de grafos, tales como "encontrar a todas las personas que conocen a alguien interesado en las tecnologías de la Web Semántica". SPARQL puede ser utilizado para consultar archivos RDF independientes, así como conjuntos de archivos RDF, ya sea cargados en la memoria por el motor de consulta SPARQL o a través del uso de un almacén de triplas compatible con SPARQL.

El acceso a los datos se tiene a través de lo que se conoce como *endpoint* o punto final SPARQL –una aplicación que responde a consultas SPARQL– y gracias a este elemento se puede tener acceso a grandes cantidades de datos RDF y se puede acceder también a bases datos que no sean de triplas. Es una buena práctica para los proveedores de conjuntos de datos exponer y proporcionar la URL de su punto final de SPARQL para permitir el acceso a sus datos de forma programada o a través de una interfaz Web [6].

SPARQL tiene una sintaxis similar a Turtle y ofrece cuatro formas de consulta que pueden ser utilizadas para ejecutar diferentes tipos de consultas:

- SELECT, utilizado para recuperar información basada en un patrón particular.
- CONSTRUCT, consulta con respuesta en forma de grafo RDF y que puede ser empleada como un servicio de traducción de datos RDF (entre diferentes ontologías).
- ASK, consulta con respuesta booleana. Sirve para identificar si un patrón de consulta particular puede ser coincidente en el grafo RDF consultado.
- DESCRIBE, utilizado para identificar todas las triplas relacionadas con el recurso particular que debe ser descrito.

Una particularidad y ventaja de SPARQL es que las consultas no se limitan al predicado sino que se puede preguntar sobre cualquier parte de los datos (sujeto, predicado u objeto) e incluso sobre la ontología (por ejemplo, por propiedades). Esto es muy útil para explorar conjuntos de datos no conocidos. Además, el orden en el que se escriben las triplas no supone diferencia en el grafo de datos RDF, es decir, no afecta a los resultados de la consulta, pero sí puede afectar a la velocidad de procesamiento de las mismas.

Una consulta básica de tipo *SELECT* tiene dos partes: las palabras clave o variables (precedidas por un símbolo de interrogación) y el patrón de consulta (*WHERE*). El motor de consulta encuentra todas las coincidencias para el patrón de consulta en los datos y devuelve todos los valores para los que la palabra

clave ha coincidido. Un ejemplo de consulta SPARQL [7] para averiguar los árboles cuyas anotaciones han sido creados por el IFN es la siguiente:

```
SELECT ?tree
WHERE{
    ?tree  sta:hasAnnotation ?annotation .
    ?annotation :createdBy :ifn .
}
```

Existen numerosas palabras clave en SPARQL. Por ejemplo, para filtrar los resultados se tienen *DISTINCT* y *FILTER*; para coincidencias opcionales *OPTIONAL*; para introducir un subgrafo y encontrar triplas que no estén en este *UNSAID*... Además, SPARQL proporciona diferentes modificadores (como *ORDER* o *LIMIT*) para organizar los diversos resultados.

Aunque SPARQL es un componente clave de la Web Semántica, en los inicios tenía algunos límites que ya han sido resueltos por el estándar SPARQL 1.1. Un ejemplo de novedad respecto a SPARQL son las funciones de agregación (*MIN, MAX, COUNT, GROUP BY...*) o las consultas transitivas mediante los operadores ("*", "+") [7].

A nivel de implementaciones concretas, hay una gran cantidad de motores SPARQL. Uno de los más utilizados es OpenLink Virtuoso [49], ya que es el motor de consultas RDF detrás de la DBpedia [46] (proyecto para la extracción de datos de Wikipedia y que propone una versión de Web Semántica). Es una plataforma híbrida de almacén de triplas SPARQL 1.1 y *middleware* basada en RDBMS (Sistema de Gestión de Bases de Datos Relacionales). Está desarrollado en C y está disponible tanto a través de código abierto como de licencia comercial.

Las posibilidades de SPARQL y la complejidad del estándar no se reducen únicamente a los aspectos tratados en este apartado, donde únicamente se ha pretendido dar una visión muy resumida a modo de introducción.

2.2.5 Arquitectura de aplicaciones

La arquitectura de las aplicaciones de la Web Semántica tiene una serie de componentes comunes que distan de lo que sería una aplicación convencional con una base de datos. Los principales componentes son [7]:

- *Parseador/Serializador* RDF: se encarga de leer el texto en uno o más formatos e interpretarlo como triplas en el modelo de datos RDF (parseador) o el proceso inverso; convertir de triplas a formas serializadas (serializador).
- Convertidores y *scrapers*: convierten otras fuentes de datos como pueden ser tablas, bases de datos... a RDF.
- Almacén RDF: base de datos donde se almacenan y recuperan datos en forma de triplas. Tiene las funciones habituales de una base de datos y, adicionalmente, otras que permiten combinar información de múltiples orígenes.
- Motor de consultas RDF: posee la capacidad para recuperar información del almacén RDF de acuerdo a las consultas. El motor de consultas puede tener capacidad de inferencia, de forma que respondería no solo con triplas afirmadas sino también con las inferidas a partir de reglas de inferencia como las definidas en los estándares de RDFS y OWL.
- Aplicación: procesa los datos y accede al almacén RDF a través de consultas.

En la Figura 2.3 se muestra la arquitectura [7] de una aplicación que utiliza un almacén de triplas RDF en el *back-end*. Básicamente incluyen los componentes de parseo/serialización, almacén de triplas y motor de consultas. Existen almacenes de triplas que proporcionan una interfaz SPARQL.

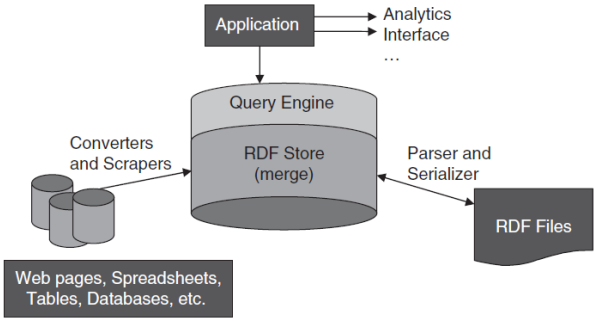


Figura 2.3: Arquitectura para aplicación con RDF [7].

2.2.6 Web Semántica Social

Dadas las limitaciones de la Web Social (ver sección 2.1.4), la Web Semántica se ajusta y resuelve varias de ellas, ya que es una plataforma muy útil a la hora de crear, reutilizar, operar y enlazar contenido dentro de un mismo sitio web social, así como con otros sitios web, que pueden ser, además, heterogéneos. Esto es debido al uso de descripciones procesables por máquina de personas y objetos y de identificadores y formatos comunes. La combinación de la Web Social y la Web Semántica –la aplicación de las tecnologías de la Web Semántica a la Web Social– tiene como resultado la Web Semántica Social [1], donde se promueven ciertas cualidades como la colaboración entre usuarios y se forma una red de contenidos y conocimientos interrelacionados y semánticamente ricos. Un esquema se puede ver en la Figura 2.4. En particular, esta integración tiene un doble enfoque: en primer lugar, la Web Semántica puede servir como una plataforma útil para enlazar y realizar operaciones en diversos datos relacionados con personas y objetos recogidos en estos sitios web; en segundo lugar, los sitios web sociales pueden servir por sí mismos como ricas fuentes de datos para las aplicaciones de la Web Semántica [1].

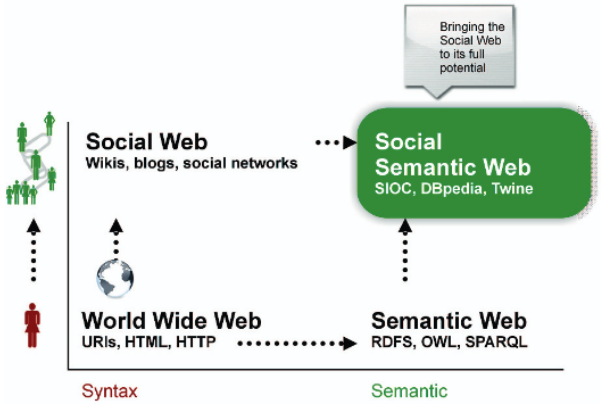


Figura 2.4: La Web Semántica Social [1].

Respecto al etiquetado y la anotación social, la inclusión de semántica en ellas tiene como resultado

anotaciones semánticas o socio-semánticas (si son generadas por usuarios de una comunidad), formadas a partir de la combinación de una anotación y esquemas de modelado formal como ontologías. Añadir semántica a las anotaciones resuelve los problemas enunciados en la sección 2.1.4 y hace de ellas una herramienta y fuente de conocimiento más valioso, ya que las etiquetas no solo tendrán significado para los humanos sino también para las máquinas. Por ejemplo, sin semántica, una anotación de posición (sean latitud y longitud) será una cadena o dos números que para una computadora o sistema *software* carece de significado; sin embargo, al añadir semántica, es decir, al usar un vocabulario bien definido, una máquina reconocerá que es una posición, de forma que se eliminan los problemas asociados al uso del lenguaje natural (sinónimos, abreviaturas, etc.). Un ejemplo concreto de ontología (vocabulario RDF básico) muy utilizada para representar latitudes, longitudes y otra información sobre recursos localizados en el espacio en el sistema de coordenadas WGS84 [50] es *Basic Geo* [51]. Se puede utilizar para anotar la posición de un árbol como se indica a continuación. De esta manera, tanto una persona como un ordenador interpretarán las coordenadas como latitud y longitud respectivamente.

```
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
sta:Tree01 geo:lat        41.56493 ;
           geo:long        -4.644281 .
```

Como se ha esbozado anteriormente, la aplicación de tecnologías semánticas en las anotaciones y recomendaciones en las redes sociales ayuda a la gestión humana de contenidos. Estos pueden ser de distinto tipo: textuales o multimedia. Por ejemplo, en el caso de imágenes, la W3C [52] hace referencia a varios vocabularios, aplicaciones y casos de uso que pueden ser utilizados para tales tareas. Una forma sencilla de hacerlo es representar metadatos relacionados con una imagen en particular (como el título, el autor, los datos de la imagen, etc.) utilizando vocabularios comunes de la Web Semántica como FOAF [48] o Dublin Core [40]. Esto proporciona entonces un medio para consultar metadatos sobre las imágenes de manera unificada. Yendo más allá, el estándar MPEG-7 (Grupo de Expertos en Imágenes en Movimiento) y sus mapeos RDF(S)/OWL asociados pueden utilizarse para representar regiones de imágenes y añadir anotaciones particulares sobre ellas. Estas anotaciones pueden combinarse con otros metadatos, por ejemplo, la modelización de que una región representa a una persona (identificada mediante el vocabulario FOAF), un lugar, etc. Esto se aplicará posteriormente en los capítulos 3 y 4.

Por último, hay que señalar las ventajas que supone aprovechar la información generada por la comunidad (datos sociales). Esta "inteligencia" colectiva se genera típicamente de forma gratuita (aunque puede ser retribuida) y puede ser, en muchos casos, más valiosa que información generada por las propias empresas. Un ejemplo concreto es Amazon; en el pasado disponían de expertos contratados para realizar críticas de libros pero, posteriormente, prescindieron de ellos al darse cuenta que los propios usuarios hacían las críticas gratis y eran más útiles para la comunidad. Además de ser ventajoso para los publicadores de datos, en el ejemplo Amazon, también lo es para los consumidores de datos, clientes, ya que estas anotaciones sociales les proporcionan un medio de expresión de sus opiniones y valoraciones.

2.3 Publicación de datos forestales

Un caso específico de publicación de datos en la Web son los datos forestales. Los volúmenes de datos que se manejan en este campo son muy elevados, ya que se necesita gran cantidad de datos para hacer gestión y planificación forestal. En la actualidad cada vez son más frecuentes las iniciativas de datos abiertos relacionadas. Estos datos se almacenan en los inventarios y mapas forestales existentes en todos los países europeos y son comunes en todo el mundo. Los inventarios siguen metodologías de muestreo para extrapolar las existencias de recursos forestales del país, mientras que los mapas (a veces incluidos en mapas más generales) se crean mediante la ortofotografía y la interpretación de la fotografía para crear información cartográfica sobre esos recursos. La metodología exacta varía de un país a otro, pero

los resultados son comparables y es necesario dedicar multitud de recursos tanto a nivel económico, como de esfuerzo y tiempo para generarlos. A pesar de considerarse datos abiertos, la mayoría de los resultados de los inventarios y mapas no suelen ser accesibles o se publican en formatos cuyas especificaciones no están a disposición del público en general.

En particular, en España el inventario (Inventario Nacional Forestal, IFN [9]) se publica en bases de datos de Microsoft Access con un esquema difícil de encontrar. Además, esta solución tecnológica no es adecuada para grandes volúmenes de datos o usuarios ni para grandes proyectos, ya que no es multiplataforma. Los mapas (Mapa Forestal de España, MFE [53]) se publican en archivos de forma SIG (Sistema de Información Geográfica) que, si bien son una norma internacional, son difíciles de utilizar fuera de la comunidad de SIG [54]. El formato de estos ficheros es *shapefile* [55], un formato de datos abierto y popular para almacenar información geográfica y de atributos y transferirlos. Sin embargo, la información que permite almacenar este formato es limitada, tanto en volumen, como en riqueza. Para almacenar información, los *shapefiles* necesitan de 3 ó más ficheros con distintas extensiones. En cada fichero componente se pueden almacenar hasta 2GB de datos. Los métodos de compresión de estos ficheros no son muy potentes y tienden a ocupar mucho más que una base de datos geométrica. Además, muchas de las herramientas que permiten formatear y crear ficheros con este formato corrompen los ficheros. Todas estas limitaciones más las asociadas a los formatos de los datos, campos y atributos (no soportan caracteres ANSI, no hay valores nulos, los campos de nombre solo pueden tener hasta 10 caracteres...), ponen de manifiesto que no es el mejor formato para publicar los datos y resulta anticuado.

Los repositorios españoles con datos forestales más destacables son el IFN [9] y el MFE [53]. Ambos contienen datos oficiales y abiertos mantenidos por el Gobierno de España.

Por un lado, en el IFN actualmente los datos más recientes son del IFN3, correspondientes al Tercer ciclo del inventario. Es un proyecto encaminado a obtener el máximo de información posible sobre la situación, régimen de propiedad y protección, naturaleza, estado legal, probable evolución y capacidad productora de todo tipo de bienes de los montes españoles. La unidad básica de trabajo es la provincia y, al ser un inventario continuo, se repiten las mismas mediciones cada 10 años, recorriéndose todo el territorio nacional en cada ciclo decenal [9]. La metodología de muestreo es la división del territorio en cuadrículas de un kilómetro cuadrado y el muestreo de los árboles dentro de esta, según su altura y distancia al centro de la celda, tomando medidas de cada árbol como la altura o el diamétro. Los datos recogidos se almacenan en varios ficheros y se publican en una base de datos SQL con un esquema *ad-hoc* y en formato propietario (Microsoft Access) [54].

Por otro lado, el MFE es la cartografía básica forestal a nivel estatal. Recoge la distribución de los ecosistemas forestales españoles. Su objetivo principal es ser la cartografía base del IFN. Por ello, análogamente al IFN, la unidad básica de trabajo es la provincia y su periodicidad decenal [53].

A nivel europeo y con el propósito de superar las limitaciones referentes a la publicación, accesibilidad y el formato de los datos, el proyecto Cross-Forest [12] nace con el objetivo de desarrollar servicios de Infraestructuras de Servicios Digitales orientados al control y predicción de incendios forestales utilizando datos abiertos enlazados. Para ello, ha sido necesaria la creación de una serie de ontologías para modelizar inventarios forestales y mapas de España y Portugal, así como posiciones y medidas. Las ontologías generadas son: *Simple Positions Ontology*, *EPSG Ontology*, *Simple Measures Ontology*, *IFN Ontology* y *MFE Ontology* [54]; cuya intención es que sean utilizables a nivel internacional para representar y publicar los datos de inventarios y mapas forestales de cualquier país en un formato abierto y estándar.

El diseño modular y la vinculación de las ontologías con otras externas busca la reutilización de estas y de los propios datos por agentes externos. En el caso concreto de España, los datos a transformar y publicar en formato RDF son los del IFN y MFE (Mapa Forestal Español). Se publican como LOD

en un almacén de triplas Virtuoso que cumple los requisitos de datos abiertos del Gobierno de España (y de Portugal), haciendo uso de las tecnologías de la Web Semántica: RDF, RDFS, OWL y SPARQL. Como resultado los datos se autodescribirán y serán interoperables, lo que permitirá que sean utilizados por el público en general y que se conecten con otros datos, ya sean datos forestales de diferentes lugares o cualquier otro tipo de datos, enriqueciendo sus posibilidades de uso. El consumo de los datos puede llevarse a cabo a través de descargas de conjuntos de datos volcados, a través del punto SPARQL o de IRIs desreferenciados [56].

En relación con lo anterior, existen una serie de aplicaciones que utilizan datos forestales con distinto alcance y objetivos. Algunas aplicaciones de interés por su aportación a este Trabajo Fin de Máster o visión similar se enumeran a continuación:

- **Forest Explorer** [10]: aplicación web de consulta de los bosques de España que permite descubrir las especies, árboles y biomasa por provincia; encontrar los árboles de parcelas cercanas; aprender las características de las especies; consultar las medidas dendrométricas de cualquier árbol o parcela; y filtrar por distintos criterios, navegando a través de un mapa con un navegador. Utiliza datos abiertos enlazados del Tercer Inventario Forestal Nacional de España(IFN3) y del Mapa Forestal Español (MFE50) generados por el proyecto Cross-Forest. Estos datos han sido convertidos a RDF, anotados con la ontología forestal desarrollada, enriquecidos con anotaciones de otras fuentes e integrados en una única fuente de datos. Se empleará como fuente de datos para la aplicación desarrollada en este trabajo y se diferencia de esta en que no permite creaciones (anotaciones) y no es social.
- **TreeTalk** [14]: aplicación web y móvil forestal social que sugiere rutas o paseos por la ciudad de Londres entre dos puntos o indicando un punto de partida. Indica árboles a lo largo de la ruta, su especie y porcentaje de población en Londres y fotos. También señala otros puntos de interés en el camino como cafeterías y permite buscar árboles en un mapa. La parte social está implementada con Twitter, embebido en la página web y que muestra *tweets* que comparten los usuarios con árboles cercanos mediante *hastags*. No es aplicable dentro del territorio nacional y no permite consultar datos de inventarios forestales.
- **Pl@ntNet** [15]: aplicación web y móvil, de tipo enciclopédico, colaborativa para la recopilación, anotación y recuperación de imágenes para la identificación de plantas de forma automática a partir de una base datos botánica. Está limitada únicamente a la temática de las imágenes, por lo que no tiene muchos puntos en común con la idea que se presenta en este trabajo.

En definitiva, la aplicación de tecnologías de la Web Semántica en el ámbito forestal permite la publicación de los datos de forma abierta y accesibles al público general, vincularlos con ontologías relacionadas con territorios geográficos y políticos o de descripción de especies. El resultado de este tipo de publicación beneficia a organismos gubernamentales y no gubernamentales, empresas y particulares [54].

2.4 Conclusiones

A lo largo de este capítulo se ha expuesto cómo la Web es un medio para el intercambio de datos en auge y con gran potencial, sin embargo no está exento de una serie de dificultades relacionadas con la publicación de los datos en ella para su posterior utilización, teniendo en cuenta las necesidades de los consumidores. Estos datos pueden tener orígenes muy variados así como restricciones. Actualmente, gran parte de los datos disponibles en línea son abiertos y muchos de ellos sociales –generados por usuarios de comunidades. Con el propósito de superar los desafíos para con los datos, surgen los datos enlazados abiertos enlazados, con el objetivo de establecer normas internacionales de la web para la publicación y difusión de datos estructurados, convirtiendo a la WWW en una base de datos mundial conocida como Web Semántica. Para poder explotar el potencial de los datos enlazados de la Web Semántica son

necesarias una serie de tecnologías y lenguajes para publicar, enlazar, consultar y modelar conjuntos de datos, así como conocer los fundamentos de ellas y las buenas prácticas recomendadas por la W3C. La Web Semántica Social aplica las tecnologías de la Web Semántica a la Web Social para mejorar la interoperabilidad de sitios web y formar una red de contenidos interrelacionados más valiosa y que supere las limitaciones previas. Además, se pone de manifiesto el valor de los contenidos y anotaciones sociales en la Web como fuente de conocimiento de calidad.

Particularizando la temática de datos en la Web a un campo concreto como es el forestal, se pueden identificar dificultades propias de publicación de datos en inventarios o mapas forestales comunes y que pueden resolverse mediante la aplicación de tecnologías de la Web Semántica. Como caso de estudio se pueden tomar los repositorios forestales gubernamentales españoles tradicionales como el IFN o el MFE y señalar sus deficiencias; así como las ventajas y potencial que tiene la transformación de estos a datos abiertos enlazados como por ejemplo la realizada por el proyecto Cross-Forest. Adicionalmente, se entrevé cómo la sinergia entre ciudadanos y administraciones públicas puede enriquecer los conjuntos de datos y se enuncian varias aplicaciones que utilizan datos forestales, por ejemplo de etiquetado de árboles, e incluyendo algunas de ellas una importante parte social.

Capítulo 3

Análisis

Una vez abordados los conceptos teóricos que sirven de base para la comprensión de las ideas y tecnologías fundamentales que se emplearán en este proyecto, es necesario contextualizar su origen, especificando el problema que se desea resolver. El presente capítulo se centrará en la fase de análisis de la ingeniería de *software* [17, cap. 2] para la construcción del *software* del *back-end* de la aplicación y las etapas de conceptualización de la fase de desarrollo de ontologías para la creación del modelo de datos [16]. Se definirá el dominio de la aplicación y se identificarán y describirán los requisitos del sistema, tanto funcionales como no funcionales, y los casos de uso mediante una descripción informal y centrándose en la parte servidora de la aplicación que se plantea como solución. Por último, se analiza el modelo de datos para el sistema, tomando el dominio como eje central e identificando los principales conceptos y cómo se relacionan.

3.1 Contexto y origen del proyecto

El origen de este proyecto tiene como inicio la detección de una necesidad no cubierta por la oferta general existente en el área forestal. Esta es la relacionada con la anotación de especies forestales arbóreas y la inexistencia de aplicaciones colaborativas sociales en España para el etiquetado de estas. Por otro lado, también se incluye la consulta de datos forestales por público no experto, sean aficionados o estudiantes, debido a la complejidad que tiene la consulta de datos en inventarios forestales.

En el ámbito de la ciencia e ingeniería forestal la cantidad de datos que se manejan es muy elevada dada la gran longevidad de los árboles en general, así como el gran volumen de masas forestales existentes. Estos conjuntos de datos se recogen de forma oficial por ingenieros forestales en inventarios y mapas forestales mantenidos por administraciones públicas. A pesar de la precisión de los datos, tienden a estar aislados y la forma en la que se almacenan –bases de datos con esquemas desconocidos o formatos dispares– hacen que su utilización no sea trivial y esté restringida únicamente a expertos, aún cuando son datos abiertos y que cualquiera puede consultar. Asimismo, los datos inventariados son limitados –lógicamente el coste de elaborar estos muestreos, medidas e inventarios es muy elevado, por lo que se establece un compromiso datos de interés/coste asumible– y se restringen a ciertas áreas o masas forestales consideradas de utilidad por diversos motivos como control de incendios o de plagas. Además, su actualización es periódica cada cierto número de años (decenal en el caso del IFN [9]) por lo que estos datos no siempre están actualizados. Del mismo modo, los datos muestreados y almacenados en los inventarios oficiales son únicamente textuales y cartográficos, pero no contienen otro tipo de contenidos que podrían tener valor como elementos multimedia tales como una galería de imágenes.

Las limitaciones referentes a la publicación de los datos en la Web y su consulta, teniendo en cuenta las recomendaciones de la W3C [2], convierten las tecnologías de datos abiertos enlazados y de la Web Semántica en candidatos ideales para facilitar la integración, accesibilidad y reutilización de los datos

forestales, permitiendo consultas más complejas a causa de la semántica subyacente y, por tanto, datos con un mayor valor y más próximos a los usuarios como han demostrado proyectos como Cross-Forest [12] y aplicaciones como el "Explorador forestal" que hace uso de estos datos abiertos enlazados [10].

En este contexto, junto con la intención de establecer un proyecto de ciencia ciudadana [3] en el que los ciudadanos colaboren de forma activa en la recolección de datos forestales, se enmarca este Trabajo Fin de Máster. Resulta atractiva la idea de sinergia entre instituciones y ciudadanos, con el establecimiento de un espacio social donde se cree, anote y comparta contenido entre una comunidad de usuarios interesados en materia forestal. De esta forma, aparte de consultar datos gubernamentales forestales, gracias a la colaboración ciudadana se podrían ampliar los datos inventariados. Este enriquecimiento de los conjuntos de datos forestales disponibles, añadiendo árboles no inventariados, como por ejemplo los pertinentes a las zonas urbanas; fotografías o datos de árboles existentes; sugerencias de actualización de datos, por ejemplo, notificar que un árbol ha sido talado; puede ser atractivo en varios sectores como el de gestión administrativa y el educativo.

Por un lado, a nivel de administración y organismos públicos una plataforma con una gran cantidad de datos forestales, integrada con inventarios oficiales, fácil de utilizar y con formatos estándares puede ayudar a una una gestión más eficiente de sus parques y jardines, por ejemplo para control de plagas o existencia de especies concretas en puntos determinados de la ciudad. Este sistema podría ser un primer punto de consulta de acceso sencillo útil en multitud de casuísticas, o si fuera necesaria más información, una vez identificado lo que se desear buscar, acceder de forma más sencilla a un inventario más completo, pero también más complejo.

Por otro lado, en el entorno educativo, una plataforma donde se fomente la participación y discusión en la identificación de especies, además de ser un punto de consulta de fácil acceso para cualquier persona, podría ser un marco excelente para plantear diversas tareas educativas como la identificación especies de árboles en los espacios verdes cercanos y animar al alumnado a la participación en alguna formación relacionada con el ámbito forestal, en la creación de contenido colaborativo y su intervención en discusiones al respecto. También se podría considerar como una fuente de información de datos completamente fiables (en el caso de datos oficiales o sociales aceptados) o en estudio (para datos sociales sin validar).

En una primera aproximación, los datos que almacenará la aplicación serán limitados y más orientados a sus posibles aplicaciones en el marco educativo, pero se plantea un diseño abierto de forma que se puedan ampliar en un futuro con datos más técnicos como mediciones dendrométricas (ya incluidas en el IFN). Para este Trabajo Fin de Máster se consideran de interés:

- La posición absoluta de los árboles en el sistema de coordenadas WGS84 [50].
- Las especies de los árboles, seleccionadas a partir de una amplia taxonomía de especies proveniente del IFN que ha sido trasladada a la ontología de "Cross-Forest"[12] y cargada en la aplicación.
- Las imágenes de los árboles o partes de estos, para construir una una galería multimedia, basada en fotografías anotadas a partir de sus metadatos más la información adicional que incluya el creador: el título, una descripción, temática de la parte del árbol que muestra (vista general, copa, rama, tronco, hoja, flor, fruto u otros) o la anotación/identificación de partes significativas en la imagen (por ejemplo, señalar un fruto en una imagen de un árbol, un nido...).

3.2 Visión general del sistema

Dentro del marco de desarrollo de la aplicación se plantea una visión general de esta (ver Figura 3.1). Consta de cuatro capas: capa de presentación, capa de aplicación, capa de acceso a datos y capa de

datos. Los usuarios finales interactuarán típicamente[1] con la aplicación a través de la capa de aplicación, que en el caso de un *front-end* de tipo aplicación web, estaría distribuida entre el navegador del usuario y el *front-end*. Estos usuarios pueden ser de dos tipos: usuarios no registrados, con capacidades únicamente de consulta; y usuarios registrados, que, además de operaciones de consulta, tienen la posibilidad de crear y anotar árboles. Dentro de los usuarios registrados existen usuarios expertos que pueden, además de crear y consultar datos, validar anotaciones de otros usuarios. Cabe apuntar que, como se señala en la Figura 3.1, este Trabajo Fin de Máster se centra en la parte servidora o *back-end* de la aplicación.

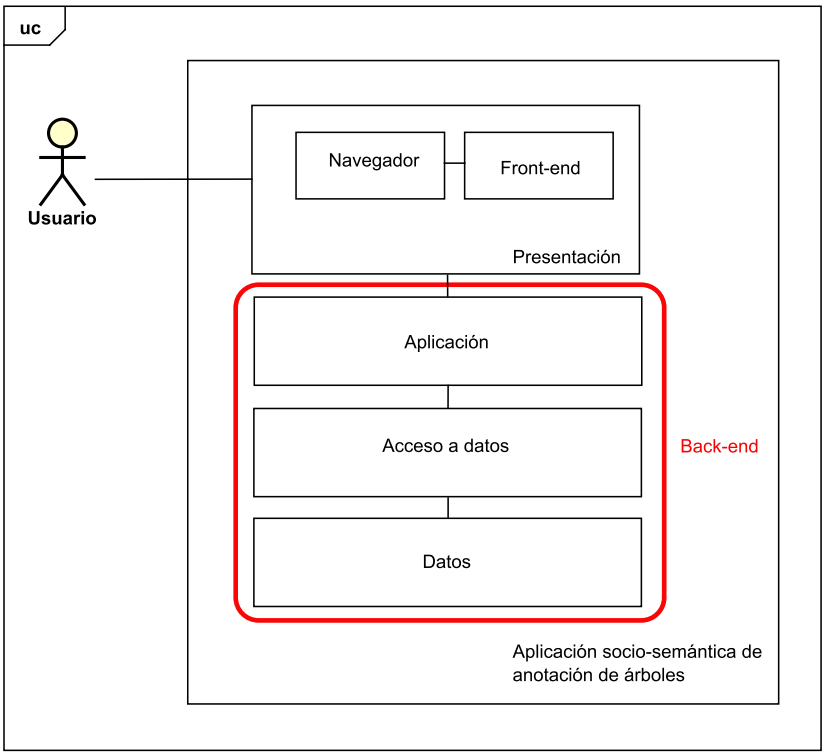


Figura 3.1: Vista general del sistema.

Desde la perspectiva del *back-end* –el análisis se centra exclusivamente en esta parte y este constituirá el sistema– incluye las capas de aplicación, acceso a datos y datos. Tiene un único actor principal: el *front-end*. Además, una de las partes más importantes del *back-end* es el modelo de datos. Será una ontología la que permita definir los términos necesarios para el sistema de manera formal e integrar datos en RDF del IFN, es decir, árboles del inventario y sus datos.

3.3 Definición de requisitos

Tras una visión general del sistema y planteando una aplicación *software* como solución se procede a la especificación de los requisitos. Un requisito [57] es una declaración abstracta de alto nivel de un servicio que el sistema debe proporcionar o una definición detallada y formal de una función del sistema.

[1]Los usuarios con conocimientos en tecnologías de la Web Semántica podrían consultar datos directamente de la capa de datos (posteriormente, almacén de triplas) si lo deseasen, aunque esto no formaría parte propiamente de la aplicación, sino que es una funcionalidad añadida que permitiría el sistema.

Existen dos grandes tipos de requisitos: funcionales, asociados a una declaración de una funcionalidad del sistema; y no funcionales, que se corresponden con restricciones que afectan a los servicios o funciones del sistema.

3.3.1 Requisitos funcionales

RF-001: El sistema deberá permitir registrar a los usuarios.

RF-002: El sistema deberá permitir solicitar un listado de todos los usuarios registrados.

RF-003: El sistema deberá permitir a cualquier usuario registrado solicitar la información de su cuenta de usuario (nombre, apellidos, nombre de usuario, email, puntos conseguidos y categoría).

RF-004: El sistema deberá permitir al usuario registrado etiquetar o anotar ubicaciones (coordenadas de latitud y longitud) de árboles.

RF-005: El sistema deberá permitir a cualquier usuario registrado etiquetar o anotar especies de árboles a partir de la taxonomía cargada en el sistema proveniente de la ontología del Cross-Forest [12].

RF-006: El sistema deberá permitir a cualquier usuario registrado añadir imágenes de árboles o partes de estos.

RF-007: El sistema deberá permitir a cualquier usuario registrado etiquetar o anotar imágenes al subirlas al sistema o sobre ya existentes.

RF-008: El sistema deberá permitir a cualquier usuario (registrado y no registrado) escoger un árbol y ver las anotaciones existentes, mostrando además para cada tipo de anotación cuál es la más popular.

RF-009: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar una lista de anotaciones creadas por un usuario concreto.

RF-010: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar y visualizar en un mapa los árboles registrados en el sistema.

RF-011: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar una lista de árboles de una especie concreta y visualizarlos en un mapa.

RF-012: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar una lista de los árboles registrados en un área concreta (entre dos latitudes y dos longitudes) y visualizarlos en un mapa.

RF-013: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar una lista de los árboles creados por usuario concreto y visualizarlos en un mapa.

RF-014: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar la información de un árbol del sistema (creador, latitud, longitud, especie, fecha de creación, anotaciones asociadas, tipo dentro de la ontología).

RF-015: El sistema deberá permitir al usuario registrado dar de alta en el sistema un nuevo árbol no existente, indicando como mínimo los datos obligatorios que se corresponden con la posición (latitud y longitud). Además, permitirá incluir campos opcionales como la especie (seleccionando una de las existentes en la taxonomía) y una imagen (anotada opcionalmente con: título, descripción y parte que se visualiza).

RF-016: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar una lista jerarquizada de la taxonomía de especies (clase, familia, género, especie) existente en el sistema.

RF-017: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar y visualizar las imágenes existentes de un árbol concreto registrado en el sistema.

RF-018: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar la información de una imagen asociada a un árbol existente en el sistema (tipo dentro de la ontología, fecha de creación, uri del fichero jpg y opcionalmente: parte del árbol que muestra (vista general, copa, rama, tronco, hoja, flor, fruto u otros), distintos metadatos exif que contenga la imagen: ancho y largo, geolocalización, fecha de captura).

RF-019: El sistema deberá permitir modificar los datos (nombre, apellidos, nombre de usuario, email) de un usuario registrado.

RF-020: El sistema deberá permitir modificar la contraseña de una cuenta de usuario registrado.

RF-021: El sistema deberá permitir eliminar a un usuario registrado.

RF-022: El sistema deberá permitir al usuario registrado opinar sobre anotaciones de otros usuarios indicando "Estoy de acuerdo" o "No estoy de acuerdo" con la información que contienen.

RF-023: El sistema deberá permitir al usuario registrado eliminar una anotación creada por él mismo.

RF-024: El sistema deberá permitir al usuario registrado supervisor eliminar anotaciones que sean falsas y alertar al creador de este hecho.

RF-025: El sistema deberá permitir a cualquier usuario registrado solicitar la información de sus logros[2], puntos que tiene y puntos que le faltan para subir de categoría.

RF-026: El sistema deberá permitir consultar a cualquier usuario registrado una lista con los puntos que han conseguido los usuarios recientemente y el motivo de ese logro.

RF-027: El sistema deberá permitir al usuario registrado publicar un logro en el tablón de logros del sistema.

RF-028: El sistema deberá permitir al usuario registrado compartir un logro en las redes sociales: Facebook, Twitter, Instagram y WhatsApp.

RF-029: El sistema deberá permitir al usuario registrado solicitar una lista de anotaciones en las que ha participado, ya sea por haberlas creado o haber opinado sobre ellas.

RF-030: El sistema deberá permitir al usuario registrado de categoría experto validar anotaciones de otros usuarios cuando la información que contienen la consideran correcta.

RF-031: El sistema deberá permitir a cualquier usuario (registrado) añadir información adicional acerca de un árbol existente en el sistema, considerados datos de interés en relación a control de plagas. Son anotaciones relativas a tratamientos recibidos por un árbol: fecha y producto/acción (por ejemplo, podar) aplicado o realizado a un árbol para su cuidado.

RF-032: El sistema deberá permitir a cualquier usuario (registrado y no registrado) solicitar información adicional acerca de un árbol existente en el sistema, considerados datos de interés en relación a control de plagas. Son anotaciones relativas a tratamientos recibidos por un árbol: fecha y producto/acción (por ejemplo, podar) aplicado o realizado a un árbol para su cuidado.

3.3.2 Requisitos no funcionales

RNF-001: El sistema deberá permitir anotaciones semánticas de árboles.

RNF-002: El sistema deberá paginar los resultados que devuelve, incluyendo hasta un máximo de 1000 individuos y proporcionando cómo acceder al resto de resultados si los hubiera.

[2]Logro: cualquier acción que proporciona puntos en la aplicación, sea añadir nuevos árboles, anotaciones o participar opinando, es decir, diciendo "Estoy de acuerdo" o "No estoy de acuerdo", a las anotaciones de otros usuarios.

RNF-003: El sistema deberá permitir el uso de posiciones absolutas y definidas en coordenadas del sistema de coordenadas WGS84 [50].

RNF-004: El sistema deberá tener registrados a los usuarios para que puedan crear, modificar o eliminar cualquier tipo de información, ya sea relativa a árboles, anotaciones, imágenes o usuarios.

RNF-005: El sistema deberá devolver representaciones de los recursos expuestos en formato JSON.

RNF-006: El sistema deberá resolver peticiones de lectura en tiempos inferiores a 1 segundo para garantizar la fluidez de la aplicación.

RNF-007: El sistema deberá resolver peticiones de creación en tiempos inferiores a 5 segundos para garantizar la fluidez de la aplicación.

RNF-008: El sistema deberá comprobar que el nombre de usuario y la contraseña son correctos para autenticar a un usuario y considerarlo usuario registrado.

RNF-009: El sistema deberá controlar que los usuarios registrados solo puedan modificar datos de usuario de su propia cuenta.

RNF-010: El sistema deberá controlar que los usuarios solo puedan visualizar los datos públicos de otros usuarios registrados.

RNF-011: El sistema deberá controlar que solo los usuarios supervisores puedan eliminar árboles y anotaciones de otros usuarios.

RNF-012: El sistema deberá establecer como información validada/aceptada las anotaciones afirmadas por expertos o entidades oficiales.

RNF-013: El sistema deberá tratar como logros recientes aquellos creados en las 24 horas anteriores al momento de la consulta de esta información.

3.4 Casos de uso

Una vez especificados los requisitos del sistema, se van identificar y documentar los casos de uso desde un punto de vista descriptivo y omitiendo los flujos alternativos y casos de error (estos aspectos se tratarán en los capítulos 4 y 5 correspondientes al diseño e implementación, respectivamente). Los casos de uso [57] describen el comportamiento del sistema al afrontar un requisito funcional que enfatiza el valor proporcionado por el sistema a sus actores, que son entidades externas tales como operadores humanos u otros sistemas.

El diagrama de casos de uso se ilustra en la Figura 3.2. Los casos de uso identificados para el sistema y una pequeña descripción informal de cada uno de ellos se incluye a continuación:

CU-001: Registrar usuario

Descripción breve. Un usuario se registra por primera vez en el sistema.

Precondición. El usuario no existe en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de registrar nuevo usuario y los datos introducidos por el usuario (nombre de usuario, contraseña, nombre completo y email) al sistema.
2. El sistema valida los datos introducidos, comprobando que el nombre de usuario no exista en el sistema, crea el usuario y notifica al *front-end* que la creación ha sido exitosa.

Postcondición. El usuario se ha creado en el sistema.

CU-002: Identificar usuario

Descripción breve. El usuario registrado se identifica en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de identificar a un usuario y los datos introducidos por el usuario (credenciales: nombre de usuario y contraseña) al sistema.
2. El sistema valida las credenciales, comprueba que el usuario existe en el sistema y notifica al *front-end* que el usuario ha sido identificado con éxito.

Postcondición. El usuario se ha autenticado correctamente en el sistema.

CU-003: Ver usuarios registrados

Descripción breve. El usuario consulta la lista de usuarios registrados en el sistema.

Precondición. Ninguna.

Descripción paso a paso.

1. El *front-end* envía la petición de consulta de los usuarios registrados al sistema.
2. El sistema proporciona los nombres de usuario de los usuarios registrados en el sistema al *front-end* .

Postcondición. Ninguna.

CU-004: Ver información de usuario registrado

Descripción breve. El usuario consulta su información almacenada en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de solicitar la información del usuario al sistema y las credenciales del usuario.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema proporciona el nombre completo, nombre de usuario y email del usuario al *front-end* .

Postcondición. Ninguna.

CU-005: Anotar árbol

Descripción breve. El usuario anota un árbol existente en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de crear una nueva anotación, las credenciales de usuario, el identificador del árbol que se desea anotar, el tipo de anotación a crear (especie, posición, imagen o información adicional) y los datos de la anotación (especie; latitud y longitud; imagen (obligatorio, el resto opcionales), título de la foto, descripción, elemento que muestra, partes a seleccionar y elementos que etiquetar; o información, respectivamente en función del tipo de anotación) al sistema.
2. El sistema comprueba que el usuario está registrado previamente en el sistema y que sus credenciales son correctas.
3. El sistema comprueba que la información recibida es correcta, es decir, que la información obligatoria ha sido recibida y que el árbol con el identificador especificado existe.
4. El sistema notifica al *front-end* que la anotación ha sido realizada correctamente.

Postcondición. El usuario registrado ha creado una anotación de un árbol en el sistema.

CU-006: Ver anotaciones

Descripción breve. El usuario consulta las anotaciones existentes en el sistema.

Precondición. Ninguna.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar las anotaciones del sistema con un creador concreto y el nombre de usuario del creador para el que se desea consultar o una anotación concreta mediante su identficador único.
2. El sistema valida que el nombre de usuario (o el identificador) introducido exista y devuelve las anotaciones existentes al *front-end* .

Postcondición. Ninguna.

CU-007: Ver árboles

Descripción breve. El usuario consulta los árboles existentes en el sistema.

Precondición. Ninguna.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar los árboles del sistema. Se pueden solicitar todos lo árboles o filtrarlos por estos criterios: localizados en un área concreta (entre dos latitudes y longitudes dadas en grados decimales), de una especie, creados por un usuario determinado, un árbol específico. La petición incluye estas restricciones si las hubiera.
2. El sistema envía una respuesta que incorpora hasta 1000 árboles filtrados según las restricciones incluidas en el paso 1 (si las hubiera) al *front-end* .

Postcondición. Ninguna.

CU-008: Registrar árbol

Descripción breve. El usuario registra un nuevo árbol en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de crear un nuevo árbol, las credenciales de usuario y los datos del árbol. Es obligatorio incluir la posición del árbol (latitud y longitud) y opcional la especie del árbol y una imagen de este. La imagen se puede anotar opcionalmente con el título de la foto, descripción y elemento que muestra. Adicionalmente, se pueden seleccionar y anotar partes de la imagen de forma opcional y etiquetar qué se muestra en estas partes.

2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema comprueba que la información recibida es correcta, es decir, que la información obligatoria ha sido recibida y que el árbol con el identificador especificado existe en el sistema.
4. El sistema notifica al *front-end* que el árbol ha sido registrado correctamente.

Postcondición. El usuario registrado ha creado un árbol en el sistema.

CU-009: Ver especies

Descripción breve. El usuario consulta la taxonomía de especies existentes en el sistema.

Precondición. Ninguna.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar las especies del sistema.
2. El sistema envía una respuesta que incorpora la taxonomía de especies (clases, familias, géneros y especies) al *front-end*.

Postcondición. Ninguna.

CU-010: Ver imagen

Descripción breve. El usuario consulta una imagen existente en el sistema.

Precondición. Ninguna.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar una imagen concreta del sistema con el identificador de esta.
2. El sistema envía una respuesta que incorpora las anotaciones existentes de la imagen y la ruta al fichero que la contiene al *front-end*.

Postcondición. Ninguna.

CU-011: Modificar perfil

Descripción breve. El usuario registrado modifica sus datos personales en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de modificar los datos del usuario, las credenciales del usuario y los datos que se desean modificar introducidos por el usuario (nombre completo y/o email y/o contraseña) al sistema.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema actualiza y modifica los datos almacenados en el sistema con la nueva información introducida por el usuario y notifica al *front-end* que el perfil del usuario ha sido modificado con éxito.

Postcondición. El usuario ha modificado sus datos en el sistema.

CU-012: Baja de usuario

Descripción breve. El usuario registrado se da de baja en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de eliminar un usuario y las credenciales del usuario al sistema.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema elimina al usuario y sus datos personales y notifica al *front-end* que el usuario ha sido borrado con éxito.

Postcondición. El usuario se ha eliminado del sistema.

CU-013: Valorar anotación

Descripción breve. El usuario califica una anotación existente en el sistema.

Precondición. El usuario está registrado en el sistema y hay al menos una anotación.

Descripción paso a paso.

1. El *front-end* envía la petición de valorar una anotación, las credenciales del usuario, la valoración (positiva o negativa, para indicar si está de acuerdo o no) y un comentario opcional al sistema.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema valida que la anotación exista en el sistema y la valoración sea correcta y notifica al *front-end* que el la valoración se ha añadido con éxito.

Postcondición. El usuario ha valorado una anotación del sistema.

CU-014: Eliminar anotación

Descripción breve. El usuario registrado elimina una anotación en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de eliminar anotación y las credenciales del usuario al sistema.
2. El sistema comprueba que el usuario está registrado en el sistema, sus credenciales son correctas y si es supervisor.
3. El sistema elimina la anotación si el creador es el usuario que desea eliminarla o independientemente del creador solo si el usuario es supervisor y notifica al *front-end* que ha sido borrada con éxito.

Postcondición. El usuario ha eliminado una anotación del sistema.

CU-015: Ver Mis Logros

Descripción breve. El usuario consulta sus logros en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar logros de un usuario y las credenciales del usuario.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema envía una respuesta que incluye todos los logros para el usuario existentes en el sistema al *front-end*.

Postcondición. Ninguna.

CU-016: Ver logros recientes

Descripción breve. El usuario consulta los logros recientes en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar logros y las credenciales del usuario.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema envía una respuesta que incluye los logros creados en las últimas 24 horas en el sistema al *front-end*.

Postcondición. Ninguna.

CU-017: Compartir logro

Descripción breve. El usuario comparte uno de sus logros en el sistema o por redes sociales.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de compartir logros, las credenciales del usuario y cómo desea compartirlo (en el tablón de la aplicación o en una red social externa: Facebook, Twitter, Instagram y WhatsApp).
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema comparte el logro como proceda según el paso 1. y notifica al *front-end* que el logro ha sido compartido con éxito.

Postcondición. El usuario ha compartido un logro.

CU-018: Ver Mi Actividad

Descripción breve. El usuario consulta su actividad reciente en el sistema.

Precondición. El usuario está registrado en el sistema.

Descripción paso a paso.

1. El *front-end* envía la petición de consultar actividad de un usuario y las credenciales del usuario.
2. El sistema comprueba que el usuario está registrado en el sistema y que sus credenciales son correctas.
3. El sistema envía al *front-end* una respuesta que incluye una lista de anotaciones creadas por el usuario o que han sido valoradas por él.

Postcondición. Ninguna.

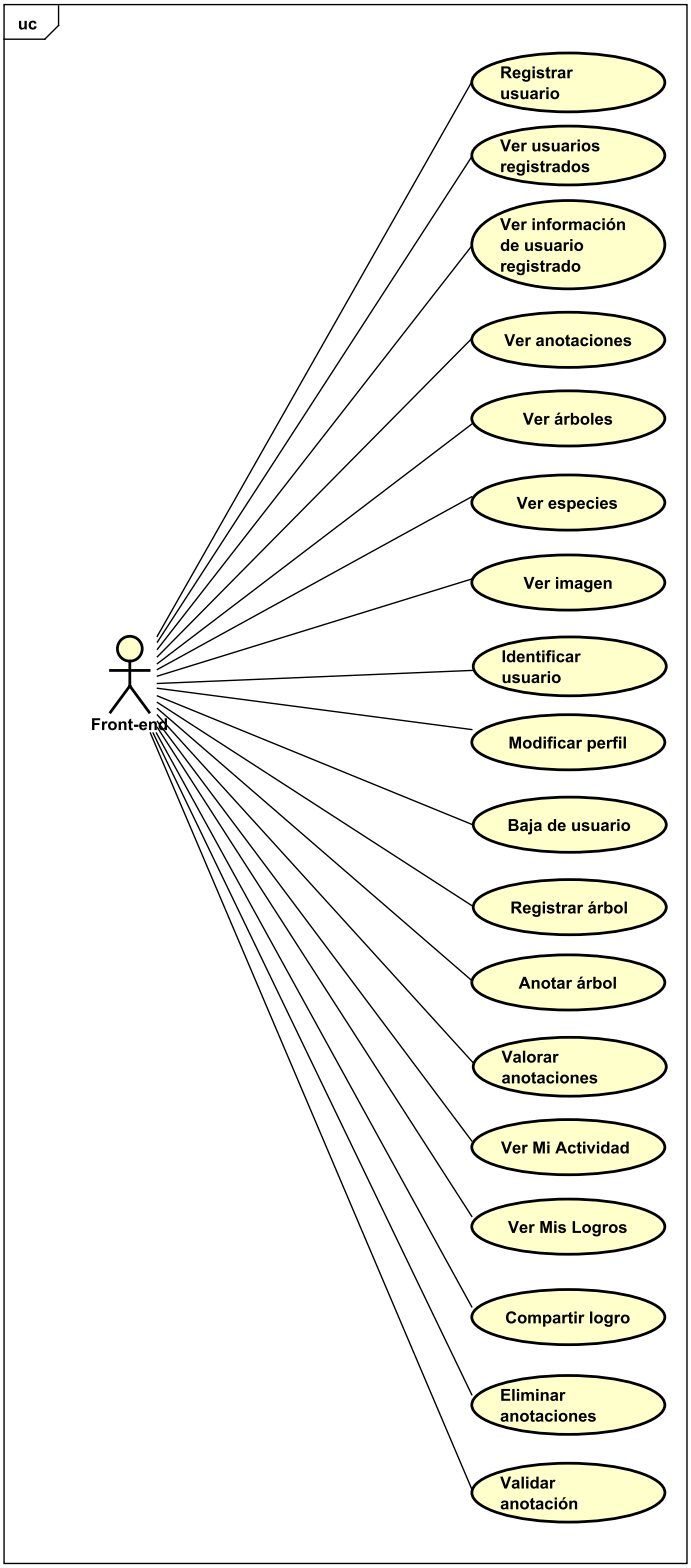


Figura 3.2: Diagrama de casos de uso del sistema.

3.5 Modelo de datos

El modelo de datos muestra cómo se representan los datos y las relaciones entre ellos, es decir, permite representar el conocimiento para un dominio específico. Para ello, se puede construir una ontología a partir de un estudio del dominio. Como se ha explicado en el capítulo 2, una ontología es un modelo formal –vocabulario con un conjunto de términos que se utilizan en un ámbito particular– que permite representar el conocimiento para un dominio específico. Aunque idealmente se debería tratar de satisfacer todos los requisitos y casos de uso previamente presentados, en este Trabajo Fin de Máster el modelo de datos se va a centrar en que incluya lo necesario para poder dar de alta árboles con anotaciones de posición, especie o imagen elaboradas por usuarios y consultar estos mismos datos de árboles del IFN. Por tanto, además se tendrán que poder dar de alta usuarios. De la misma forma, se incorpora la parte de anotación de imágenes. Los casos de uso que se tratan son los 12 primeros, del CU-001 a CU-012 inclusive.

3.5.1 Identificación de elementos del dominio

Sobre la base de los requisitos y los casos de uso de la aplicación, teniendo en cuenta el requisito RNF-001:, se pueden identificar una serie de elementos presentes en los conjuntos de datos que se utilizarán y generarán por la aplicación. En la Figura 3.3, se muestra un mapa conceptual del dominio con la intención de proporcionar una visión general de este, representando los conceptos con cajas y su nombre en el primer compartimento, las relaciones entre ellos con flechas, las propiedades de cada concepto en el segundo compartimento de las cajas y la cardinalidad con números en los extremos de las flechas. Una relación especial es la representada por las flechas huecas, hacen referencia a especializaciones de conceptos (abajo) más generales (arriba).

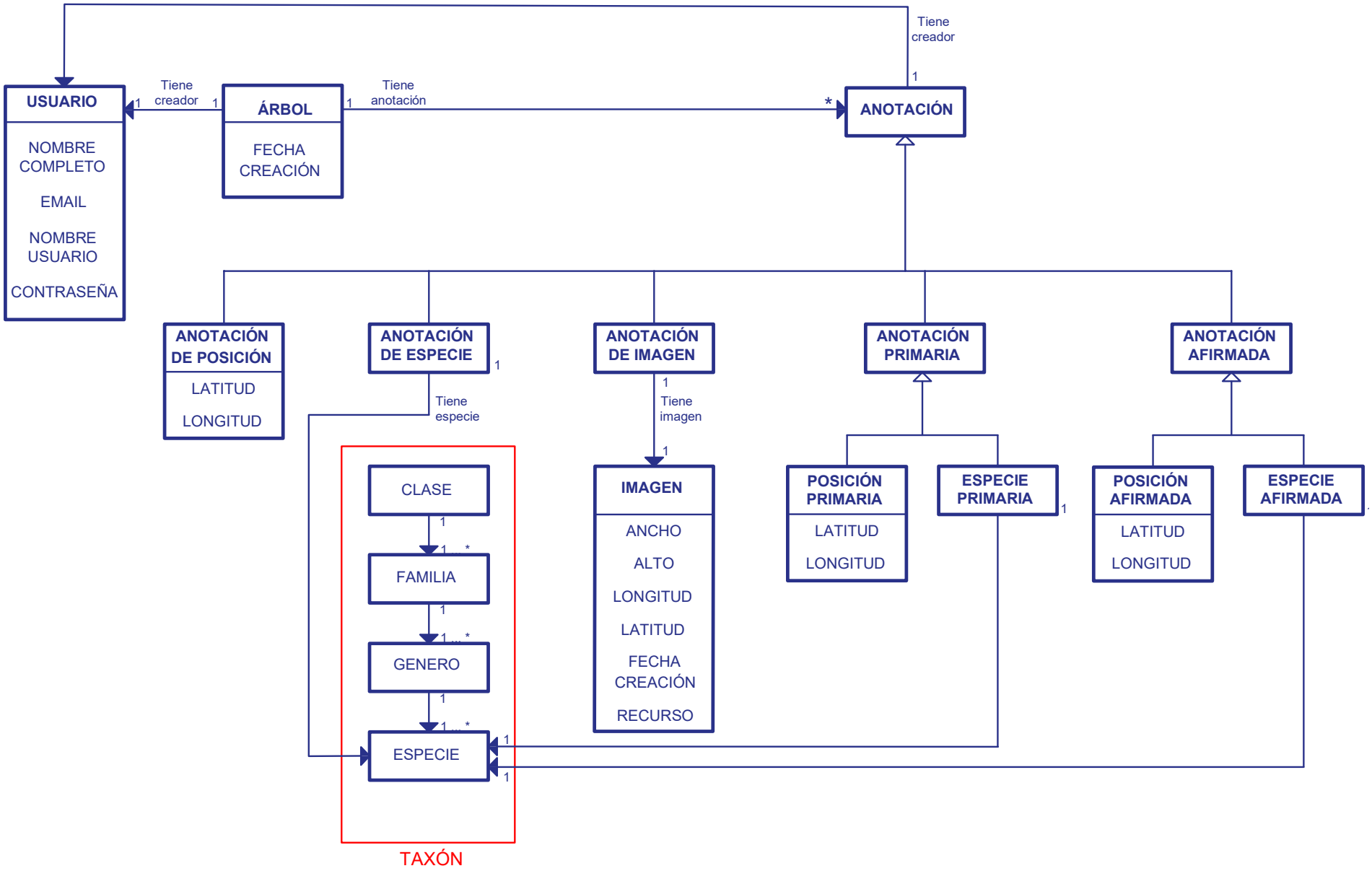


FIGURA 3.3: Mapa conceptual del dominio.

A continuación, se explican los principales conceptos del dominio identificados:

- **Usuario:** hace referencia a los usuarios registrados en la aplicación. Estos usuarios deberán tener nombre completo, nombre de usuario, email y contraseña asociados.
- **Árbol:** modela los árboles de la aplicación. Estos árboles deberán tener un creador y una fecha de creación. Podrán tener múltiples anotaciones asociadas, referentes a la posición, especie o imágenes de estos.
- **Posición:** se refiere a una posición absoluta en el sistema de coordenadas WGS84 [50] de los árboles o la geolocalización de las imágenes tomadas.
- **Taxón:** término más general que especie por definición. Cada una de las subdivisiones de la clasificación biológica, desde la especie, que se toma como unidad, hasta el filo o tipo de organización [58]. Todas las especies pertenecen a un taxón. Permite modelar la taxonomía de especies.
- **Clase:** grupo taxonómico que comprende varios órdenes de plantas con muchos caracteres comunes.
- **Familia:** taxón constituido por varios géneros naturales que poseen gran número de caracteres comunes.
- **Género:** taxón que agrupa a especies que comparten ciertos caracteres.
- **Especie:** último escalón de la taxonomía de especies forestales existente en el sistema.
- **Anotación:** contenido semántico que se añade a los árboles por un usuario registrado, es decir, información sobre los árboles que el creador considera de interés.
- **Anotación de posición:** información de la posición de un árbol dada por un usuario registrado.
- **Anotación de especie:** información de la especie de un árbol dada por un usuario registrado.
- **Anotación de imagen:** información de la imagen de un árbol o partes de este tomada por un usuario registrado.
- **Anotación primaria:** anotación que contiene la información que se considera más veraz en un momento dado sobre un árbol, por ser la más popular.
- **Posición primaria:** anotación de tipo posición más popular para un árbol en un momento dado.
- **Especie primaria:** anotación de tipo especie más popular para un árbol en un momento dado.
- **Anotación afirmada:** anotación que contiene información afirmada por usuarios expertos en temática forestal o proveniente de datos oficiales como los del IFN.
- **Posición afirmada:** anotación de tipo posición validada para un árbol.
- **Especie afirmada:** anotación de tipo especie validada para un árbol.
- **Imagen:** imagen tomada por un usuario registrado y que puede incluir metadatos.
- **Parte del árbol identificada en la foto:** posibles partes que se pueden identificar en una imagen de un árbol.
- **Hoja de árbol en foto:** identifica una hoja de un árbol en una imagen.
- **Fruto de árbol en foto:** identifica un fruto de un árbol en una imagen.

- **Tronco de árbol en foto:** identifica un tronco de un árbol en una imagen.
- **Rama de árbol en foto:** identifica una rama de un árbol en una imagen.
- **Corona de árbol en foto:** identifica una corona de un árbol en una imagen.
- **Flor de árbol en foto:** identifica una flor de un árbol en una imagen.
- **Vista general de árbol en foto:** indica que lo que se observa en la imagen es una vista general de un árbol.
- **Otra parte de árbol en foto:** indica que lo que se observa en la imagen es otro elemento de un árbol.

Las propiedades del dominio que permiten relacionar los conceptos son:

- **Creado por/Tiene creador:** relaciona un concepto con su creador.
- **Nombre completo:** nombre y apellidos de un usuario registrado.
- **Email:** correo electrónico de un usuario registrado.
- **Nombre de usuario:** *nickname* de un usuario registrado con el que se identifica en el sistema.
- **Contraseña:** cadena de caracteres secreta de un usuario registrado que, junto con el nombre de usuario, le permite identificar su identidad en el sistema.
- **Fecha de creación:** marca temporal de la creación de un elemento en el sistema.
- **Tiene anotación:** permite relacionar un árbol con una anotación.
- **Subclase de:** se utiliza para relacionar un concepto general del sistema con otros conceptos más específicos.
- **Tiene anotación de posición:** relaciona un árbol con una anotación de posición.
- **Latitud:** permite indicar las coordenadas asociadas a la latitud de una posición absoluta en el sistema de coordenadas WGS84 [50].
- **Longitud:** permite indicar las coordenadas asociadas a la longitud de una posición absoluta en el sistema de coordenadas WGS84 [50].
- **Tiene anotación de especie:** relaciona un árbol con una anotación de especie.
- **Tiene especie**: relaciona una anotación de especie con la especie concreta de la taxonomía de especies del sistema.
- **Tiene taxón**: relaciona una anotación de especie con el elemento adecuado (clase, familia, género, especie) de la taxonomía de especies del sistema.
- **Tiene anotación de imagen:** relaciona un árbol con una anotación de imagen.
- **Tiene imagen**: relaciona una anotación de imagen con la imagen concreta del sistema.
- **Ancho de imagen**: tamaño en píxeles de la anchura de una imagen.
- **Alto de imagen**:tamaño en píxeles de la altura de una imagen.
- **Recurso**: relaciona la imagen anotada con el fichero imagen que contiene la propia fotografía.

- **Tiene anotación primaria:** relaciona un árbol con una anotación primaria.
- **Tiene posición primaria:** relaciona un árbol con una anotación primaria de posición.
- **Tiene especie primaria:** relaciona un árbol con una anotación primaria de especie.
- **Tiene anotación afirmada:** relaciona un árbol con una anotación afirmada.
- **Tiene posición afirmada:** relaciona un árbol con una anotación afirmada de posición.
- **Tiene especie afirmada:** relaciona un árbol con una anotación afirmada de especie.

3.6 Conclusiones

El análisis ha permitido contextualizar de manera más concreta el proyecto, proporcionar una visión general de la aplicación propuesta, identificar a los usuarios que interactuarán con ella y establecer claramente la frontera del *back-end* (el sistema), acotando las capas y elementos que pertenecen a este, así como el actor que interactuará con el sistema. Desde esta perspectiva se han definido los requisitos, tanto funcionales como no funcionales, los casos de uso, en los que se refleja la lógica que debe recoger el sistema y el análisis del modelo de datos. Para el modelo de datos ha sido necesario realizar un estudio del dominio, mediante la identificación de los principales conceptos presentes en este y las relaciones entre ellos.

Capítulo 4

Diseño del back-end de anotación semántica de árboles

En este capítulo se plantea un diseño para satisfacer los requisitos descritos en el capítulo 3. Se divide en cinco partes, ilustrando en primer lugar la arquitectura general del sistema y sus componentes. En segundo lugar, se realiza el diseño de la ontología basado en el dominio partiendo del análisis del modelo de datos del capítulo 3. Para ello, se estudian y analizan ontologías populares existentes que puedan ser de utilidad para el sistema que se pretende diseñar (tanto en el ámbito forestal como otras más generales). A continuación, se propone una ontología que satisface las necesidades de la aplicación, así como los IRIs de esta. De acuerdo a las ontologías anteriores, se presentan los datos a exponer por el sistema con ejemplos representativos de las principales entidades del sistema. Posteriormente, se explica la integración de datos de fuentes externas en el sistema, en concreto del IFN. Finalmente, se desarrolla el diseño de la API REST que se va a exponer a los clientes, tomando como herramienta principal el modelo de recursos y definiendo los patrones de los IRIs e identificadores únicos de los recursos, así como elaborando una documentación detallada de toda la API, imprescindible para los clientes y desarrolladores.

4.1 Arquitectura del sistema

El sistema que se propone en este Trabajo Fin de Máster es el servidor de una aplicación web basada en Web Semántica Social con datos abiertos enlazados con la que puedan interaccionar distintos clientes a través de una API utilizando redes de comunicaciones y protocolos de la Web. Esta interacción de los usuarios con el sistema está pensada para que sea, principalmente, de forma "indirecta" a través de un *front-end* como puede ser de una aplicación web o una *app* móvil. Sin embargo, podría llegar a darse de forma "directa", por ejemplo, a través de un navegador o peticiones cURL. Adicionalmente, al tratarse de LOD, se ofrece a los usuarios la posibilidad de consultar los datos directamente, sin hacer uso de la API, mediante consultas SPARQL al *endpoint* del almacén de triplas. Esta posibilidad puede resultar de utilidad para usuarios familiarizados con la Web Semántica.

El diseño que se plantea se ilustra en la Figura 4.1 y consta de una serie de componentes, los cuales serán alojados/instalados en una máquina virtual creada en la nube de cómputo Openstack del grupo GSIC/EMIC.

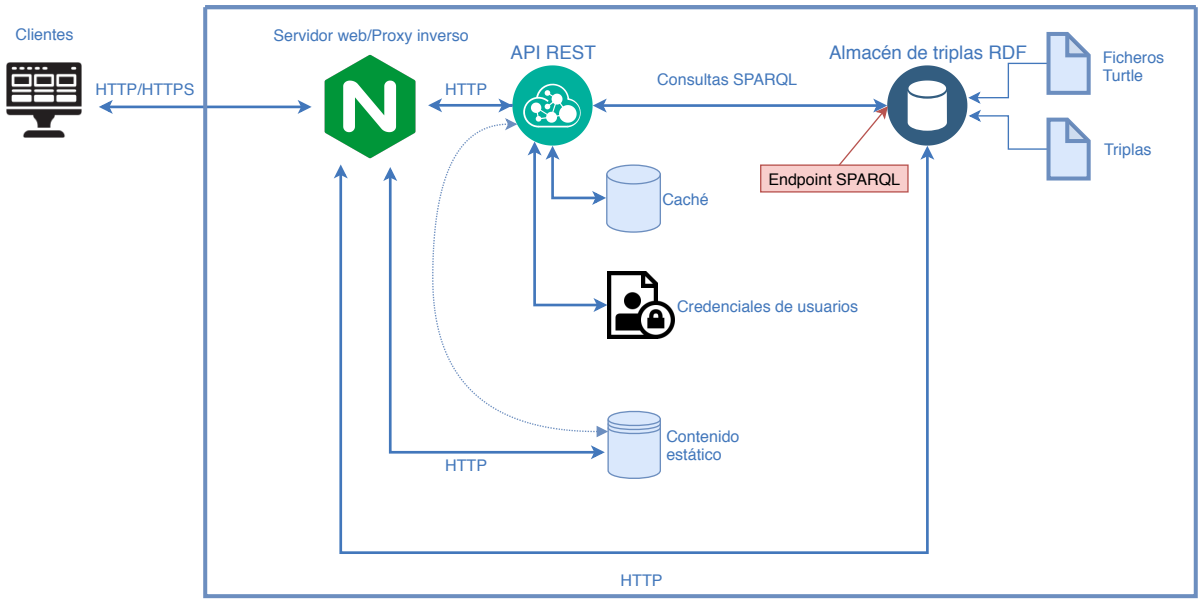


Figura 4.1: Arquitectura del *back-end*.

Los componentes son los siguientes:

- **Almacén de triplas RDF:** es el lugar donde se almacenarán los datos o en el argot de la Web Semántica, las triplas y la ontología diseñada. Los datos exportados de otras aplicaciones y la ontología se cargarán a través de ficheros Turtle con extensión *.ttl* y los datos creados por el sistema se insertarán mediante consultas SPARQL de inserción. Asimismo, este componente soportará SPARQL para las consultas, integrará el motor de consultas y ofrecerá un servicio como interfaz (*endpoint* SPARQL).

- **Caché:** es un componente que almacenará los resultados de las peticiones que se realicen a la API en memoria, de forma que solicitudes futuras de esos datos se puedan atender con mayor rapidez, reduciendo el número de consultas SPARQL.

- **Credenciales de usuarios:** con el objetivo de reducir la complejidad, en vez de utilizar una base de datos para la gestión de usuarios se empleará un fichero en texto plano que almacena las credenciales de los usuarios del sistema, es decir, los nombres de usuario y los *hashes* asociados a las contraseñas. Será necesario para la autenticación y gestión de usuarios.

- **API:** se encargará de atender y recibir peticiones y actuar como *proxy* entre los usuarios (clientes) y el almacén de triplas, traduciendo peticiones HTTP en consultas SPARQL. Además de enmascarar la complejidad subyacente a las tecnologías propias de la Web Semántica a las potenciales aplicaciones clientes que se integren ofreciendo una interfaz uniforme, con métodos bien definidos y sistemática para consultar y crear nuevos datos mediante HTTP, también se ocupará de la gestión de usuarios y el uso y mantenimiento de una caché para la optimización del rendimiento y reducir el número de transacciones con el almacén de triplas.

- **Contenido estático:** contendrá los ficheros de las imágenes que formarán la galería de imágenes de la aplicación y almacenará las nuevas imágenes que vayan subiendo los usuarios.

- **Servidor web y *proxy* inverso** [22]: realizará las funciones de servir contenido estático y de servidor intermediario o de paso que reenviará las solicitudes de contenido de múltiples clientes a diferentes servidores. En este caso, será el responsable de redirigir las peticiones a la API o al *endpoint* SPARQL según corresponda (si se implementase un cliente web también estaría detrás de este componente). Proporciona un nivel adicional de abstracción y control para asegurar el flujo fluido del tráfico de la red entre los clientes y el servidor y se encargará de la encriptación SSL [59] para quitar carga al servidor web de la API y aumentar su rendimiento.

4.2 Ontología

A partir de los conceptos y relaciones identificados en el dominio en capítulo anterior (ver 3.5), se diseñan las clases y propiedades de la ontología y sus jerarquías, respectivamente. Adicionalmente, se pueden incluir restricciones y reglas. En un segundo paso, se deben de intentar reutilizar ontologías conocidas como recomienda la bibliografía [1, 7].

En este Trabajo Fin de Máster se va a presentar una primera versión de la ontología, la cual no se diseñará por completo para satisfacer todos los requisitos descritos en la sección 3.3, pero será ampliable en un futuro si se desea. En concreto, se va a centrar en que incluya lo necesario para poder dar de alta árboles con anotaciones de posición, especie o imagen elaboradas por usuarios y consultar estos mismos datos de árboles del IFN. Por tanto, además se tendrán que poder dar de alta usuarios. De la misma forma, se incorpora la parte de anotación de imágenes.

Aunque la mayor parte de los términos anteriores son propios del dominio y, por tanto muy particulares, existen otros generales como: imagen y sus metadatos Exif [60], vocabulario relacionado con las personas y sus datos personales u otros metadatos de recursos como pueden ser los definidos por DCMI [40] para especificar creadores, fechas... A partir de la identificación de conceptos del análisis de datos 3.5 se puede realizar una etapa de búsqueda de ontologías con términos similares a los identificados.

4.2.1 Estudio de ontologías conocidas para el dominio

A lo largo de este apartado se van a presentar, en primer lugar, una serie de ontologías forestales que pueden reutilizarse para algunos términos necesarios específicos de este sector. Después se analizarán varias ontologías bien conocidas populares de Internet para los términos más genéricos.

4.2.1.1 Ontologías forestales: Cross-Forest

Dado que el proyecto se centra en el desarrollo de una aplicación de anotación de árboles y, por tanto, se enmarca en un contexto forestal, se procede al estudio de las ontologías generadas por el proyecto Cross-Forest [12] teniendo presente el objetivo de poder importar árboles del IFN en esta aplicación. Sin embargo, este estudio no va a ser exhaustivo debido a la complejidad y extensión que tienen las ontologías y se va a centrar principalmente en la parte necesaria para la ulterior importación de datos. Concretamente, son esenciales para el presente trabajo las clases, propiedades e individuos relacionados con los árboles, posiciones, taxonomía de especies (clase, familia, género, especie) y provincias, teniendo en cuenta los casos de uso y requisitos definidos en la etapa de análisis (capítulo 3).

Las ontologías empleadas para modelar el conjunto de datos IFN3 junto con MFE50 y sus subontologías están documentadas [54] y para explorarlas se pueden realizar consultas sobre el *endpoint* SPARQL. Se van a comentar brevemente las 5 ontologías creadas de forma modular que se enunciaron en el apartado 2.3. Las ontologías son [54]:

- *Simple Positions*: para describir las posiciones físicas de entidades. Estas posiciones pueden ser absolutas (utilizando cualquier sistema de referencia de coordenadas) o relativas (desde una posición de referencia), ya que ambos tipos se utilizan en el dominio forestal.
- *EPSG*: es una transformación del conjunto de datos EPSG (*Geodetic Parameter Dataser*) [61] a OWL y contiene la descripción de todos los sistemas de referencia de coordenadas mantenidos por la Asociación de Productores de Petróleo y Gas (IOSP). Esta ontología puede ser utilizada junto con la ontología *Simple Positions* para describir las posiciones geográficas.
- *Simple Measures*: contiene el vocabulario para describir una medida y sus unidades (referente a las medidas de los árboles).
- *IFN*: para representar los datos del Inventario Forestal Nacional de España. En su estado actual, permite representar parcelas, árboles, y sus datos (como especies, altura, DAP (Diámetro a la Altura del Pecho o DBH en inglés, *Diameter at Breast Height*), volumen, etc.).
- *MFE*: vocabulario para representar los datos del Mapa Forestal Español. En su estado actual permite representar los datos completos del MFE-50 [53].

Atendiendo a lo anterior, con la ontología del IFN se representan árboles, parcelas y sus diferentes medidas respectivamente de acuerdo a la Figura 4.2. Como se explicó en el capítulo 2, algunas triplas extraídas de la representación gráfica en formato Turtle son:

```
ifn:Tree ifn:hasHeight ifn:Height ;
         ifn:hasDBH2 ifn:DBH2 .
...
```

Del mismo modo, el taxón representa en esta ontología como se ilustra en la Figura 4.3. Cabe señalar de la taxonomía que la especie es la clasificación más específica modelada en la ontología para un árbol.

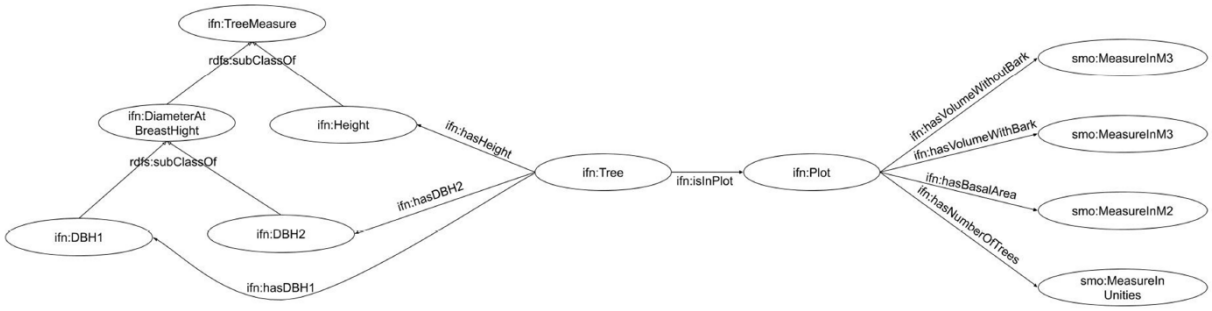


Figura 4.2: Representación de árboles y parcelas en la ontología IFN.

Para este Trabajo Fin de Máster, se van a reutilizar las clases y propiedades de la taxonomía de especies, ya que permiten modelar todas las especies recogidas en el IFN. Estas ontologías han sido realizadas por expertos en la materia [12] y ya están siendo utilizadas por otras aplicaciones como [10],

Figura 4.3: Representación del taxón en la ontología IFN.

por lo que su reutilización es adecuada en consonancia con los principios de los datos abiertos enlazados. Además de las medidas, los posicionamientos son especialmente importantes en el ámbito forestal pero no se van a tratar sus representaciones en las ontologías debido a que, por la complejidad subyacente, quedan fuera del marco de este trabajo. Para este propósito se va a utilizar la ontología *Basic Geo* [51] como se explica en el siguiente apartado.

4.2.1.2 Otras ontologías

Posteriormente, con el objetivo de modelar el resto de términos presentes en los datos se han empleado las herramientas y catálogos *Linked Open Vocabularies* (LOV) [62] y *Schema.org* [63] para buscar ontologías con términos que se desean modelar y poder reutilizarlas. Para el caso de anotación de imágenes en Web Semántica se ha seguido la documentación de la W3C [52]. Como resultado de este proceso de investigación se seleccionan las siguientes ontologías bien conocidas para su uso en los conjuntos de datos generados por la aplicación:

- FOAF (*Friend Of a Friend*) [48]: define un diccionario de términos relacionados con las personas, aunque también describen grupos o documentos (incluso metadatos de imágenes). Principalmente sus términos se agrupan en 3 categorías: central, en relación a características de las personas y grupos sociales que son independientes del tiempo y la tecnología y que pueden ser usados para describir información básica sobre las personas; Web Social, para describir recursos en este área como cuentas de Internet, *blogs*, *emails* y otras actividades basadas en la web; y utilidades de LD, ciertos términos de "demostración" del ámbito de LD. Esta ontología se reutiliza para los usuarios registrados del sistema (personas) y las imágenes.
- *Dublin Core Metadata Initiative* (DCMI) *Metadata Terms* [40]: vocabulario para expresar términos de metadatos de Dublin Core en RDF para su uso como LD. Estos metadatos son estándares y se utilizan con múltiples fines para describir todo tipo de recursos digitales (vídeo, imágenes, páginas web...) o físicos (libros, CDs...) y anotaciones (fecha de creación, creador...). Se reutiliza para los metadatos de las imágenes, asociar los creadores de los recursos y sus fechas de creación.

- Exif [60]: vocabulario para describir los datos de una imagen en formato Exif incluyendo todas las etiquetas de Exif 2.2 en formato RDF. Se va emplear para especificar el ancho y alto de las imágenes en píxeles.
- *Basic Geo (WGS84 lat/long) Vocabulary* [51]: se trata de un vocabulario básico de RDF que proporciona un espacio de nombres para representar latitud, longitud y otra información sobre elementos situadas en el espacio, utilizando WGS84 como dato de referencia. Se reutiliza para posicionar los árboles y las imágenes.

Adicionalmente, para anotar partes de imágenes se toma como ejemplo [64] y se emplea, a mayores de las ontologías y vocabularios especificados anteriormente, el vocabulario de descripción de imágenes de Jim Ley que permite definir zonas rectangulares de las imágenes mediante puntos.

4.2.2 Creación de una ontología para la anotación socio-semántica de árboles

La creación de ontologías consiste en definir todos los componentes de la ontología a través de un lenguaje de definición de ontología. Por lo general, se lleva a cabo en dos etapas [1]: una etapa informal, en la que la ontología se esboza utilizando descripciones en lenguaje natural o alguna técnica de diagramas y una etapa formal, en la que la ontología se codifica en un lenguaje formal de representación de conocimientos que es computable por máquina (por ejemplo, RDFS u OWL). También pueden utilizarse diferentes herramientas para su creación y edición (por ejemplo, Protégé [26] incluye interfaz gráfica) o visualización (WebVOWL [27]). La ontología se propone en inglés para facilitar su reutilización.

4.2.2.1 Etapa informal

A partir de los términos identificados para la aplicación en la subsección 3.5.1 y tras el estudio de las ontologías a reutilizar se establecen las clases que se requieren. Las clases modeladas así como las relaciones subclase/superclase se muestran en el diagrama de la Figura 4.4. Por otro lado, las propiedades que se definen se incluyen en la Figura 4.5. Combinando todos los elementos se obtiene una visión general de la ontología ilustrada en la Figura 4.6.

Respecto a las clases, en la Figura 4.4 se muestran con sus nombres en inglés, que posteriormente dan lugar a las IRIs. Del dominio, los requisitos, casos de uso de la aplicación y los criterios establecidos por [7]; se identifican 5 clases principales (recordando el objetivo principal de la aplicación: anotar árboles incluyendo posiciones, imágenes y especies): `TreePartPhoto` (Parte del árbol de la foto), `Tree` (Árbol), `Annotation` (Anotación), `Taxon` (Taxón) e `Image` (Imagen). Para la taxonomía de especies, como se va a reutilizar por completo la definida en las ontologías del Cross-Forest, no es necesario definir nuevas clases, se utilizan las de la ontología IFN explicada en 4.2.1.1 y las representaciones de los datos en ella (ver Figura 4.3).

En relación a las imágenes se define una especialización (subclase) de la clase `Image` de la ontología de FOAF [48], con el objetivo de añadir propiedades e información a mayores de las que establece el vocabulario de FOAF. El resto de clases se determinan en la ontología propia por ser nuevas o requerir conceptos exclusivos (como en el caso de la clase `Tree`, que no se reutiliza de la ontología IFN de Cross-Forest). En este caso, la clase `Tree` no se especializa de las IFN porque no se desea, al menos en este punto, que los árboles creados socialmente sean también individuos de la clase `Tree` del IFN. Además

de emplearse para modelar los árboles (serán individuos de la clase), contendrá información acerca de cuándo se ha dado de alta el árbol en el sistema y por quién, así como las distintas anotaciones que los usuarios hayan creado sobre él. Para la anotación de imágenes, como clases nuevas a definir son las relacionadas con `TreePartPhoto`, incluyendo sus subclases. Se emplean para clasificar e indicar qué elemento/s se retratan en las imágenes. Principalmente una imagen de un árbol puede mostrar, una vista general (`GeneralViewPhoto`), una hoja (`LeafPhoto`), rama (`BranchPhoto`), corona (`CrownPhoto`), tronco (`TrunkPhoto`), flor (`FlowerPhoto`), fruto (`FruitPhoto`) u otra parte (`OtherPartPhoto`).

Respecto a las anotaciones, se modela una superclase `Annotation` (Anotación) y especializaciones de la misma para los diferentes tipos de anotaciones que se desean realizar (de imagen, especie y posición; `ImagenAnnotation` (Anotación de imagen), `SpeciesAnnotation` (Anotación de especie) y `PositionAnnotation` (Anotación de posición), respectivamente). Adicionalmente, como un árbol puede tener distintas anotaciones de un mismo tipo y en todo momento habrá una más popular que las demás respecto a ubicación del árbol y especie (se recuerda que es una aplicación social y los usuarios pueden dar su opinión e identificar la especie de un árbol no es trivial), se define una clase `PrimaryAnnotation` (Anotación Primaria). Como subclases de esta se encuentran las anotaciones que se consideran válidas en un momento dado (por ser las más votadas, es decir, las mejor valoradas) de posición, `PrimaryPosition` (Posición primaria), y de especie, `PrimarySpecies` (Especie primaria). En el caso de que una anotación haya sido aceptada por un experto o sean datos oficiales (como por ejemplo del IFN), las anotaciones serán de tipo afirmado, `AssertedAnnotation` (Anotación afirmada), y pueden ser relativas a la especie o a la posición, `AssertedSpecies` (Especie afirmada) y `AssertedPosition` (Posición afirmada).

Las propiedades son las relaciones binarias que tendrán lugar entre individuos. Para que los datos tengan un significado deben estar relacionados y se utilizarán múltiples propiedades de los vocabularios bien conocidos estudiados previamente además de propiedades nuevas definidas en la ontología. Básicamente las propiedades propias que se requieren están relacionadas con las anotaciones y la jerarquía de clases diseñada. Un árbol puede tener una o varias anotaciones de cada tipo de los definidos en las clases, lo que resulta en propiedades como “tiene anotación”, “tiene anotación de imagen”...(ver Figura 4.5). Para las imágenes se establece una subpropiedad de FOAF `depiction`, `resource` (recurso), para enlazar el recurso imagen con el fichero *.jpg* correspondiente y para relacionar la imagen con la anotación de imagen se necesita otra propiedad, `hasImage` (tiene imagen). Relacionado con las especies, las anotaciones de especie se enlazarán con individuos de la clase Taxón del IFN, para ello se define la propiedad `hasTaxon`. En este orden de ideas, varias de propiedades estarán restringidas (recursos a los que aplica la propiedad y valores válidos de la propiedad, es decir, se establecen el dominio y rango respectivamente), como se detallará en la etapa formal. Adicionalmente se podrían definir las propiedades inversas, restricciones de cardinalidad, etc. para tener una ontología más completa.

Figura 4.4: Clases de la ontología propia para la anotación socio-semántica de árboles.

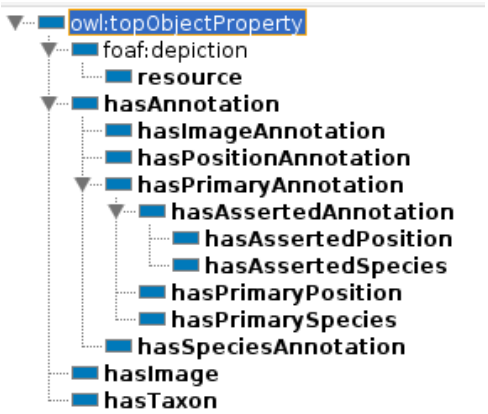


Figura 4.5: Propiedades de la ontología propia para la anotación socio-semántica de árboles.

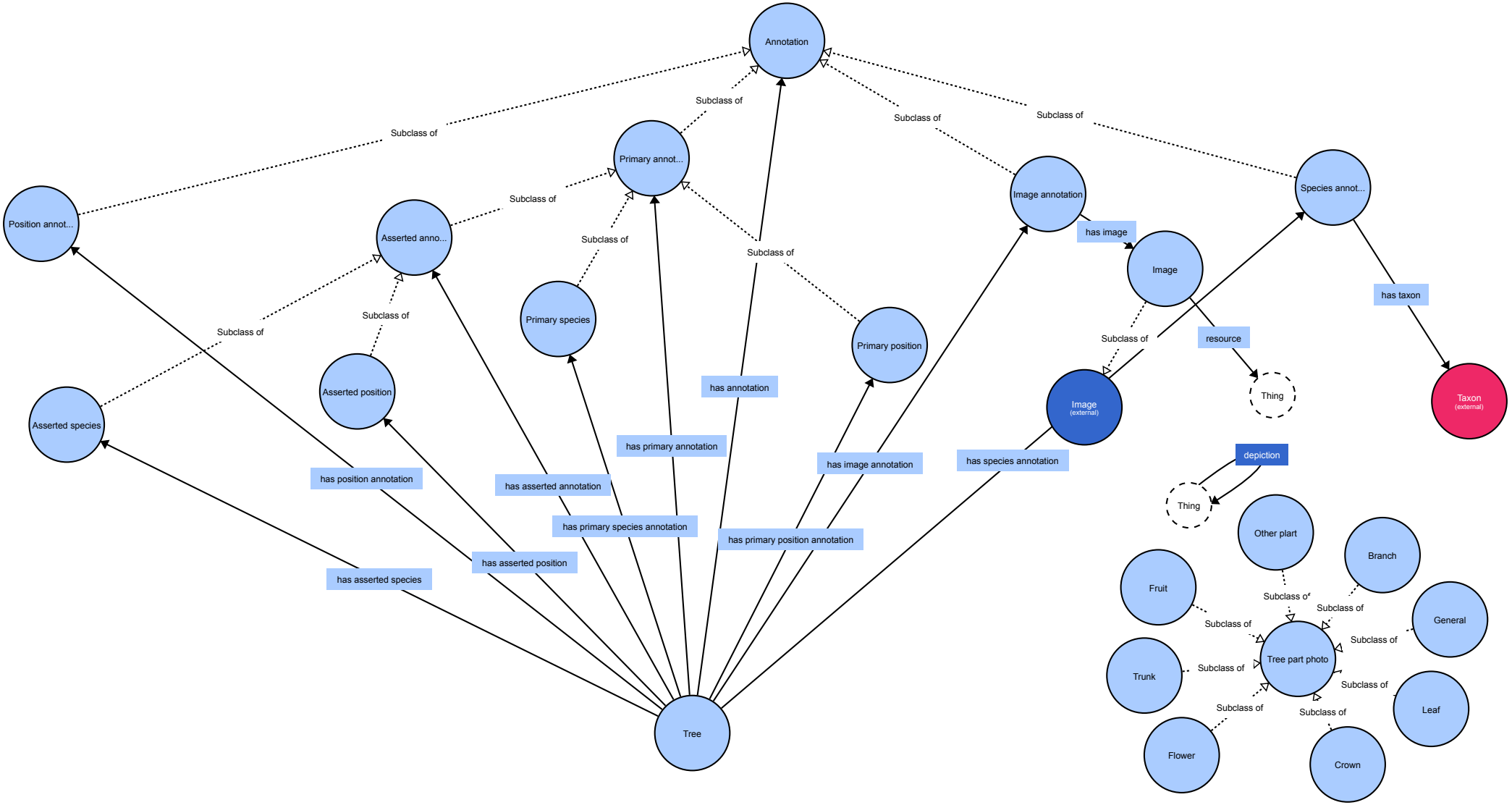


FIGURA 4.6: Visualización de la ontología desarrollada con WebVOWL.

4.2.2.2 Etapa formal

Una vez identificados todos los elementos, ya sean clases, propiedades, así como las relaciones entre ellos, se aborda la codificación de la ontología propiamente dicha. Como resultado de la etapa anterior se han identificado 21 clases y 13 propiedades de tipo *Object Property*. Se utiliza fundamentalmente el lenguaje de ontología RDFS, en el cual se describen clases y propiedades especiales para la descripción de las clases y propiedades (`rdfs:Class` y `rdf:Property`), se establecen individuos de una clase (`rdf:type` o `a`), axiomas de clase (`rdfs:subClassOf`), constructores de propiedades y axiomas (`rdfs:subPropertyOf, rdfs:domain` y `rdfs:range`) y propiedades de anotación de los recursos (`rdfs:label, rdfs:comment` y `rdfs:isDefinedBy`). Se incluye vocabulario de OWL para la cabecera de la ontología (`owl:Ontology`), definición de clases y propiedades (`owl:Class` y `owl:ObjectProperty`), importación de ontologías (`owl:imports`) y para la información de versión (`owl:versionInfo` y `owl:ObjectProperty`). El formato elegido para la serialización de la ontología es Turtle por las razones expuestas en el apartado 2.2.1.

La ontología completa se adjunta en el Apéndice A y se ha denominado STA (*Social Tree Annotation*). Su IRI es `http://timber.gsic.uva.es/sta/ontology/` y su *qname* es `sta:`, definido como `@prefix sta: <http://timber.gsic.uva.es/sta/ontology/>`. Primeramente se indican todos los espacios de nombres (*namespaces*) que se utilizan a lo largo del documento. A continuación se define la ontología y se anota añadiendo información (metadatos) sobre el creador, versión, fecha de creación, comentarios y etiquetas en dos idiomas (castellano e inglés). En la siguiente sección se definen todas las clases especificadas en la primera etapa de diseño y se anotan con sus etiquetas y comentarios relativos a cada una de ellas en los dos idiomas indicados anteriormente. Asimismo, se establece la jerarquía de subclases y se especifica que han sido definidas en esta ontología. La última sección incluye las propiedades, que en este caso son todas de tipo *Object Property*. Se define cada una de ellas con RDFS y OWL, se establece la jerarquía, es decir, se indica su propiedad padre si la tuviese, se anotan y se restringe su dominio y rango si fuera necesario. Por ejemplo, la propiedad `hasPrimarySpecies`, tiene como dominio un recurso de tipo `Tree` y como rango un recurso de tipo `PrimarySpecies`. Individuos y propiedades *Datatype Property* no se definen en este fichero ya que no son propiamente elementos de la ontología diseñada, sino que se corresponden con datos de los conjuntos de datos creados/importados o propiedades reutilizadas de otras ontologías y vocabularios.

4.2.3 Diseño de IRIs

En relación al diseño de los IRIs definidos en la aplicación, tanto en la ontología, como en los datos se han realizado siguiendo las buenas prácticas [2]. Los IRIs se establecen bajo los patrones mostrados:

- Para los elementos de la ontología:

  `http://timber.gsic.uva.es/{module}/ontology/{name},`

  siendo `module` *"sta"* en todos los casos, ya que no se han definido más módulos en esta etapa y `name` se sustituye por la clase o propiedad concreta de la ontología.

- Para los datos:

  `http://timber.gsic.uva.es/{module}/data/{type}/{name},`

análogo al caso anterior. Los detalles se explicarán en la sección 4.5, debido a que está íntimamente ligado al diseño de la API porque los datos se exponen a través de esta.

Con objeto de clarificar el diseño de los IRIs, se exponen varios ejemplos en la Tabla 4.1.

Elemento	Clase	Individuo (ejemplo)
Árbol	http://timber.gsic.uva.es/sta/ontology/Tree	`http://timber.gsic.uva.es/sta/data/tree/47-0003-A-1-1`
Anotación de especie	http://timber.gsic.uva.es/sta/ontology/SpeciesAnnotation	`http://timber.gsic.uva.es/sta/data/annotation/s-20200621-4fea2_d44e2`
Imagen	http://timber.gsic.uva.es/sta/ontology/Image	`http://timber.gsic.uva.es/sta/data/image/20200617-4dd87-3272e`

Tabla 4.1: Ejemplos de IRIs para los recursos de la ontología y los datos.

4.3 Datos expuestos

En esta sección se van a tratar los datos generados por los usuarios y los importados del IFN utilizando como modelo de datos la ontología descrita en la sección anterior 4.2. Estos datos se guardan en el almacén de triplas y se representan en la ontología creada y en las propuestas para su reutilización. Los datos generados e importados en el sistema son tipo LOD y básicamente están relacionados con árboles, sus anotaciones y usuarios. Para ilustrar los conjuntos de datos que se han diseñado y cómo se pueden representar cada uno de ellos en las ontologías se van a emplear ejemplos representativos. En la Tabla 4.2 se definen las IRIs completas de los prefijos que se van a emplear en los ejemplos ulteriores.

Prefijos de espacios de nombres	IRIs
`sta`	`http://timber.gsic.uva.es/sta/ontology/`
`staTree`	`http://timber.gsic.uva.es/sta/data/tree/`
`staUser`	`http://timber.gsic.uva.es/sta/data/user/`
`staAnnotation`	`http://timber.gsic.uva.es/sta/data/annotation/`
`staImage`	`http://timber.gsic.uva.es/sta/data/image/`
`ifn`	`http://crossforest.eu/ifn/ontology/`

Tabla 4.2: Prefijos de los espacios de nombres en formato *qnames* y la expansión en IRIs completas.

4.3.1 Ejemplos de representación de datos en las ontologías

4.3.1.1 Árbol

En la figura 4.7, se muestra la representación gráfica correspondiente a las triplas de un árbol en el modelo de datos del sistema. El árbol es un individuo de la clase `Tree` de la ontología STA y tiene asociados: un creador, una anotación de tipo imagen, una anotación primaria de posición y una fecha de creación.

4.3.1.2 Usuario

Un usuario es un recurso de tipo `Person` de la ontología FOAF. Tiene varios valores asignados mediante propiedades de esta misma ontología: el nombre completo, un nombre de usuario o *nickname* y

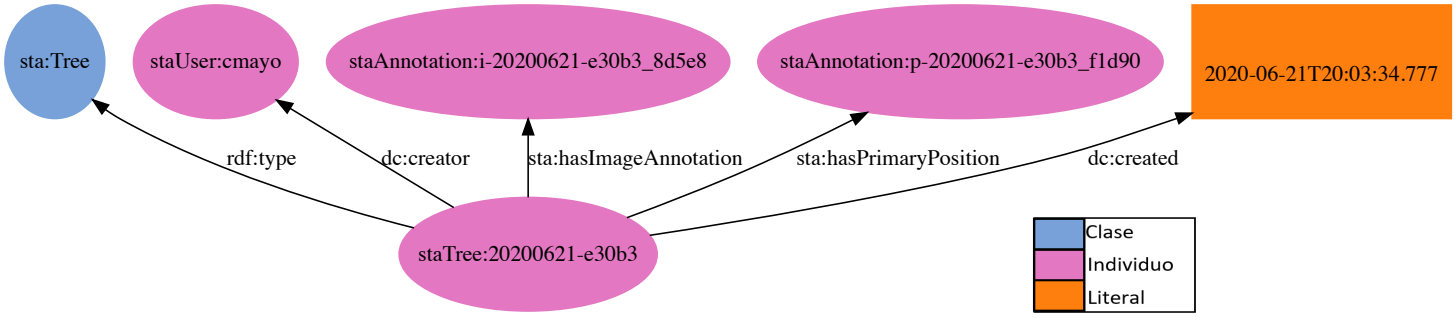


Figura 4.7: Representación de un árbol en las ontologías seleccionadas/diseñada.

un correo electrónico. Su representación se visualiza en la Figura 4.8.

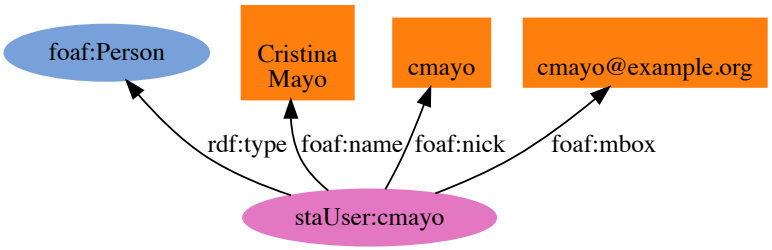
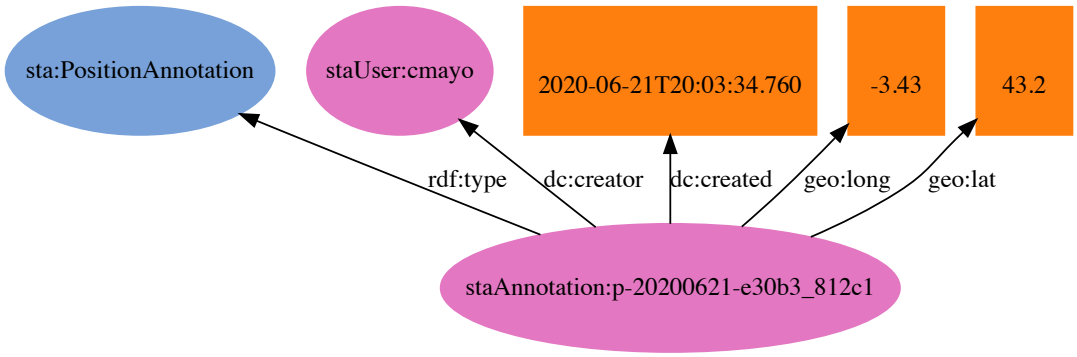


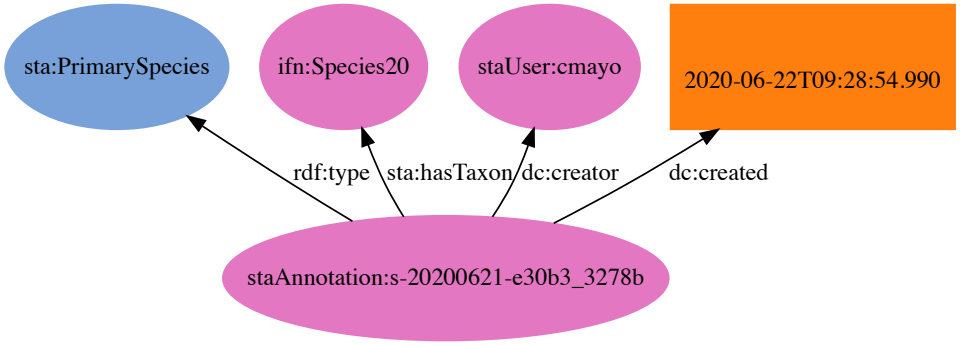
Figura 4.8: Representación de un usuario en las ontologías seleccionadas para el modelo de datos.

4.3.1.3 Anotaciones de posición, especie e imagen

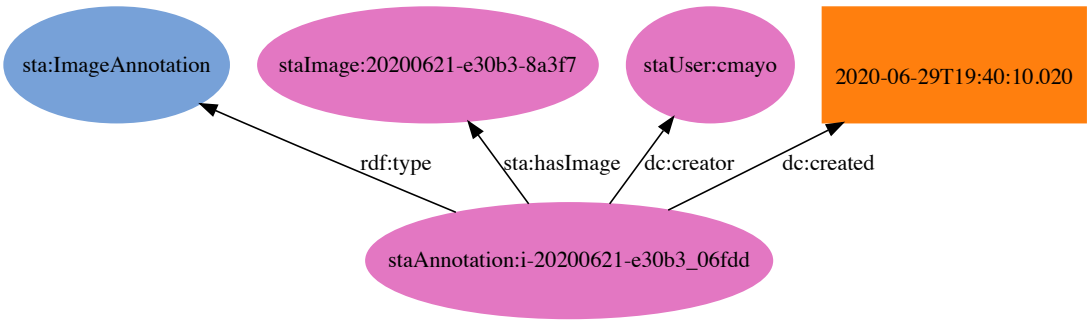
Las anotaciones que se pueden realizar sobre un árbol se agrupan en tres grandes tipos: posición, especie e imagen. Las de tipo especie y posición pueden ser de múltiples tipos de acuerdo a la ontología diseñada (Figura 4.4). Hay especializaciones para cada una de ellas como primarias o afirmadas. En la Figura 4.9 se muestran tres ejemplos: una anotación de posición de un árbol con su fecha de creación y su creador, además de la latitud y longitud (Figura 4.9a); una anotación de especie de tipo primaria, similar a la anterior pero con la especie de la taxonomía del Cross-Forest asignada (Figura 4.9b); y una anotación de imagen (Figura 4.9c) análoga a las anteriores pero relacionada con un recurso de tipo imagen.



(a) Representación de una anotación de posición de un árbol en las ontologías seleccionadas/diseñada.



(b) Representación de una anotación de especie primaria de un árbol en las ontologías seleccionadas/diseñada.



(c) Representación de una anotación de imagen un árbol en las ontologías seleccionadas/diseñada.

Figura 4.9: Representación de anotaciones de posición, especie e imagen en el modelo de datos.

4.3.1.4 Anotación de una imagen y sus partes

Las ontologías creadas y seleccionadas van a permitir que se puedan seleccionar regiones de las imágenes y asociarlas a partes de un árbol. En el siguiente ejemplo se muestra una imagen de varios manzanos y se seleccionan una manzana y una hoja concretas. Se anota la imagen en general como individuo de la clase `Image` de la ontología creada. Se clasifica como una vista general (`GeneralViewPhoto`). Se establece en la ontología Dublin Core [40] el tipo de recurso, el título, la fecha de creación, la descripción, el creador y el formato del recurso. Con la ontología de Exif [60] se anotan las dimensiones en píxeles (ancho y alto). Con *Basic Geo* se geolocaliza la imagen (latitud y longitud). Con la ontología de Jim Ley [64] se definen las regiones de la imagen, qué muestra cada una de ellas y los puntos que las definen. Se ilustran las triplas que se acaban de explicar referentes a las anotaciones RDF correspondientes a la imagen mostrada en la Figura 4.10 en forma de gráfica en la Figura 4.11. En formato serializado Turtle se indican en el código a continuación:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:        <http://www.w3.org/2001/XMLSchema#> .
@prefix exif: <http://www.w3.org/2003/12/exif/ns#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sta: <http://timber.gsic.uva.es/sta/ontology/> .
@prefix image: <http://jibbering.com/vocabs/image/#> .

<http://timber.gsic.uva.es/sta/data/image/20200621-d3c3c-e9d18>
    a sta:Image, sta:GeneralViewPhoto ;
    dc:type dc:Image ;
    dc:title  "Manzano" ;
    dc:created "2019-12-09T17:19:01"^^xsd:date ;
    dc:description "La imagen muestra un árbol frutal de tipo manzano" ;
    dc:creator <http://timber.gsic.uva.es/sta/data/user/cmayo> ;
    dc:format "image/jpeg";
    sta:resource <http://timber.gsic.uva.es/data/images/20200621-d3c3c-e9d18.jpg> ;
    exif:imageLength "600" ; # píxeles horizontal
    exif:imageWidth "450" ;  # píxeles vertical
    geo:lat 42.261251 ;
    geo:long -2.879998 ;
    image:hasPart <http://timber.gsic.uva.es/sta/data/image/20200621-d3c3c-e9d18/part/001> ;
    image:hasPart <http://timber.gsic.uva.es/sta/data/image/20200621-d3c3c-e9d18/part/002> .

# Defino 2 partes identificadas en la imagen:
<http://timber.gsic.uva.es/sta/data/image/20200621-d3c3c-e9d18/part/001> a image:Rectangle ;
        image:depicts sta:FruitPhoto ;
        image:points "92,186 156,257" ;
        dc:title "Apple" .

<http://timber.gsic.uva.es/sta/data/image/20200621-d3c3c-e9d18/part/002> a image:Rectangle ;
        image:depicts sta:LeafPhoto ;
        image:points "135,382 171,432" ;
        dc:title "Leaf" .
```

Figura 4.10: Imagen de un árbol (manzano) anotada con selección de un fruto y una hoja

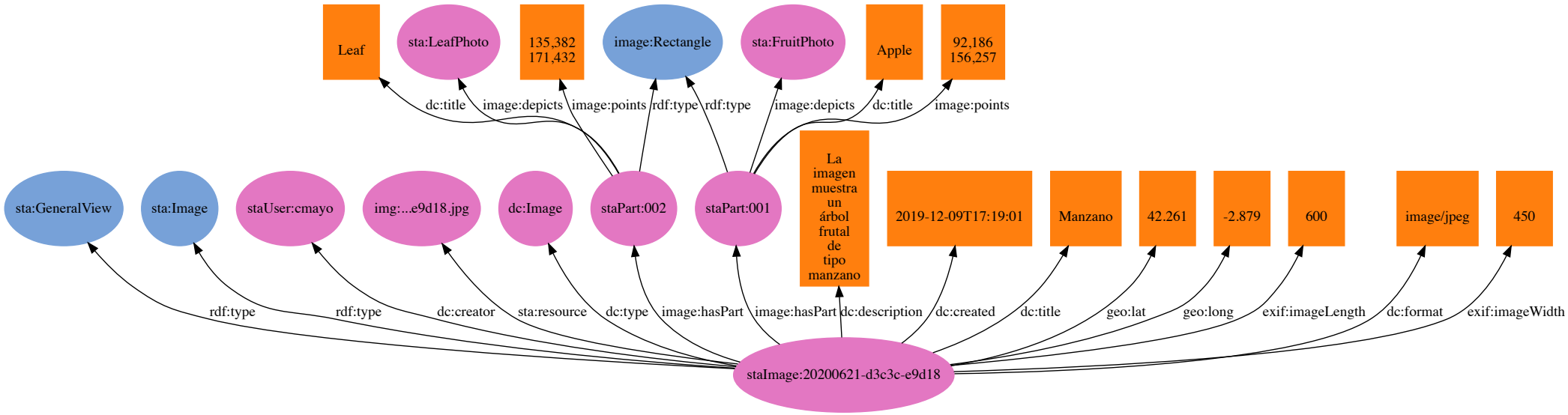


FIGURA 4.11: Representación de triplas de una imagen de un árbol completamente anotada semánticamente.

4.4 Integración de datos externos (IFN)

Un problema típico de las aplicaciones sociales es lo que se conoce como *cold-start* [65] y hace referencia a que en el inicio de una aplicación no hay datos y esto desmotiva a los potenciales usuarios. Una solución a este problema es importar los datos de árboles validados por expertos del IFN. Gracias a las ontologías desarrolladas y la extracción y conversión de datos a RDF de los datos del IFN por el Cross-Forest [12] resulta bastante inmediato –comparado con el esfuerzo que supondría esta integración sin la existencia de estas herramientas (ver sección 2.3)– la exportación de los datos del IFN en formato RDF. Esta facilidad de integración es una de las principales ventajas de la Web Semántica. Además, como se ha comentado anteriormente, el objetivo de los datos abiertos es que estén disponibles sin restricciones para aquel que desee utilizarlos. Por tanto, esta integración tiene una doble función. Por un lado, puede resolver el problema *cold-start* en el lanzamiento de la aplicación. Por otro lado, se ofrece la aplicación como un punto de acceso sencillo al inventario forestal donde se puedan consultar los datos más básicos de los árboles –localización y especie, en una primera etapa de diseño del sistema–, dejando a un lado las complejidades derivadas de los grandes conjuntos de datos, las bases de datos y las tecnologías de la Web Semántica. La influencia de esta integración sobre el diseño de la ontología y los datos no es especialmente restrictiva, pero es adecuado mostrar el efecto en los datos.

En primer lugar, para que sea posible la clasificación de los árboles mediante la taxonomía de especies del Cross-Forest [54], es necesaria la importación en el sistema de los ficheros *.ttl* que contienen los datos referentes a las clases, familias, géneros y especies así como los distintos nombres (científicos y vulgares) de las especies en inglés y en español. Es decir, el diseño de la taxonomía de especies se omite prácticamente por completo (solo se añade una propiedad a la ontología diseñada, `hasTaxon` (tiene taxón)) y se delega en la reutilización del modelo generado por el proyecto Cross-Forest, diseñado por expertos en materia forestal. Estos datos están enriquecidos además con otras fuentes como la DBpedia lo que hace que sean aún más valiosos. Una vez realizada esta carga de los ficheros en el almacén de triplas propio, ya se podrá hacer uso de la taxonomía de especies.

Posteriormente, teniendo en cuenta el estudio de la sección 4.2.1.1, se generan una o varias consultas con la finalidad de ejecutarlas sobre el *endpoint* SPARQL del "Explorador Forestal" y recuperar uno o varios grafos RDF con los datos deseados adaptados a la ontología de la aplicación desarrollada en este Trabajo Fin de Máster. Los datos exportados se cargarán en el almacén de triplas del sistema.

Como demostración de esta integración de datos exitosa, se generarán los datos de árboles con la ontología diseñada a partir de los datos del IFN de la provincia de Valladolid publicados en Cross-Forest [12] siguiendo el proceso anteriormente descrito. Los individuos son de la clase `http://crossforest.eu/ifn/ontology/Tree` de la ontología IFN y de STA y se enlaza cada uno de ellos con el individuo definido en Cross-Forest mediante la propiedad `owl:sameAs` ya que, aunque tengan distintos IRIs, hacen referencia al mismo recurso. Se establece como creador un "usuario" ficticio (http://crossforest.eu/ifn/ontology/) que hará referencia al IFN y todos los árboles tendrán una posición y una especie afirmadas y primarias, ya que son datos completamente fiables y oficiales. La consulta diseñada para alcanzar esta meta es la siguiente:

```
PREFIX ifn: <http://crossforest.eu/ifn/ontology/>
PREFIX spo: <http://crossforest.eu/position/ontology/>
PREFIX crs: <http://epsg.w3id.org/data/crs/>
```

```
PREFIX axis: <http://epsg.w3id.org/ontology/axis/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX sp: <http://vocab.linkeddata.es/datosabiertos/def/sector-publico/territorio#>
PREFIX province: <http://crossforest.eu/ifn/data/province/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX sta: <http://timber.gsic.uva.es/sta/ontology/>

CONSTRUCT {
  ?miuri a ifn:Tree , sta:Tree ;
         owl:sameAs ?tree ;
         dc:creator ifn: ;
         sta:hasAssertedPosition ?positionAnnotation ;
         sta:hasPrimaryPosition ?positionAnnotation ;
         sta:hasAssertedSpecies ?speciesAnnotation ;
         sta:hasPrimarySpecies ?speciesAnnotation .

  ?positionAnnotation a sta:AssertedPosition , sta:PrimaryPosition .
  ?positionAnnotation geo:long ?axis2 .
  ?positionAnnotation geo:lat ?axis1 .
  ?positionAnnotation dc:creator ifn: .

  ?speciesAnnotation a sta:AssertedSpecies , sta:PrimarySpecies .
  ?speciesAnnotation sta:hasTaxon ?taxon .
  ?speciesAnnotation dc:creator ifn: .
}
WHERE {
  ?tree a ifn:Tree .
  ?tree spo:hasPosition ?position .
  ?tree a ?taxon .
  ?taxon a/rdfs:subClassOf* ifn:Taxon .
  ?tree ifn:isInPlot ?plot .

  ?position spo:hasCoordinateReferenceSystem crs:4326 .
  ?position axis:1 ?axis1 .
  ?position axis:2 ?axis2 .

  ?plot sp:provincia province:47 .

  BIND (IRI(REPLACE(str(?tree),"http://crossforest.eu/ifn/data/tree/",
  "http://timber.gsic.uva.es/sta/data/annotation/p")) as ?positionAnnotation)

  BIND (IRI(REPLACE(str(?tree),"http://crossforest.eu/ifn/data/tree/",
  "http://timber.gsic.uva.es/sta/data/annotation/s")) as ?speciesAnnotation)

  BIND (IRI(REPLACE(str(?tree),"http://crossforest.eu/ifn/data/tree/",
  "http://timber.gsic.uva.es/sta/data/tree/")) as ?miuri)
}
```

Sobre la consulta anterior, en primer lugar, se definen los prefijos de los espacios de nombres para los *qnames*. A continuación, se especifica una consulta SPARQL [21] de tipo `CONSTRUCT` para que devuelva los datos en forma de grafo RDF y se da forma al grafo. En la primera parte del `CONSTRUCT` las sentencias establecen que cada árbol pertenece a las dos clases `Tree` de las dos ontologías, se enlazan con las IRIs de los árboles del Cross-Forest y se les asocian posiciones afirmadas y primarias de especie y

posición (indican que son datos validados por expertos). La segunda y tercera parte definen los individuos anotación pertenecientes a la clase que les corresponda y se les asignan, mediante las propiedades correspondientes, los valores de posición (`geo:lat` y `geo:long`) y de especie (`sta:hasTaxon`), además del creador (`dc:creator`). La cláusula `WHERE` establece el grafo patrón con el que comparar el grafo de datos y obtener datos correctos. Se busca recuperar los árboles del IFN, por tanto individuos de la clase `ifn:Tree`, su posición (aunque solo interesan la latitud y longitud) y especie. Las especies se recuperan de la taxonomía de especies establecida en la jerarquía de clases de `ifn:Taxon`. Por tanto, se puede anotar a un árbol con cualquier elemento de la taxonomía, no únicamente especie. Los árboles están en parcelas (`ifn:isInPlot`) y estas pertenecen a una provincia (propiedad `sp:provincia`). Se desea obtener los árboles de la provincia de Valladolid, por lo que se recuperarán los que sus parcelas estén en (`province:47`). Finalmente, los métodos `BIND` junto con `IRI` y `REPLACE` se utilizan para construir las IRIs deseadas de las anotaciones que se van a crear y los árboles generados, reemplazando ciertas IRIs por las especificadas y asegurándose de que sean IRIs.

El resultado es un fichero en formato Turtle (*.ttl*) con triplas que definen los datos relativos a los árboles y las anotaciones de posición y especie así como sus relaciones. Un ejemplo para un árbol concreto del IFN representado gráficamente se visualiza en la Figura 4.12, donde se puede observar que las triplas forman un grafo como el definido en el `CONSTRUCT` de la consulta anterior.

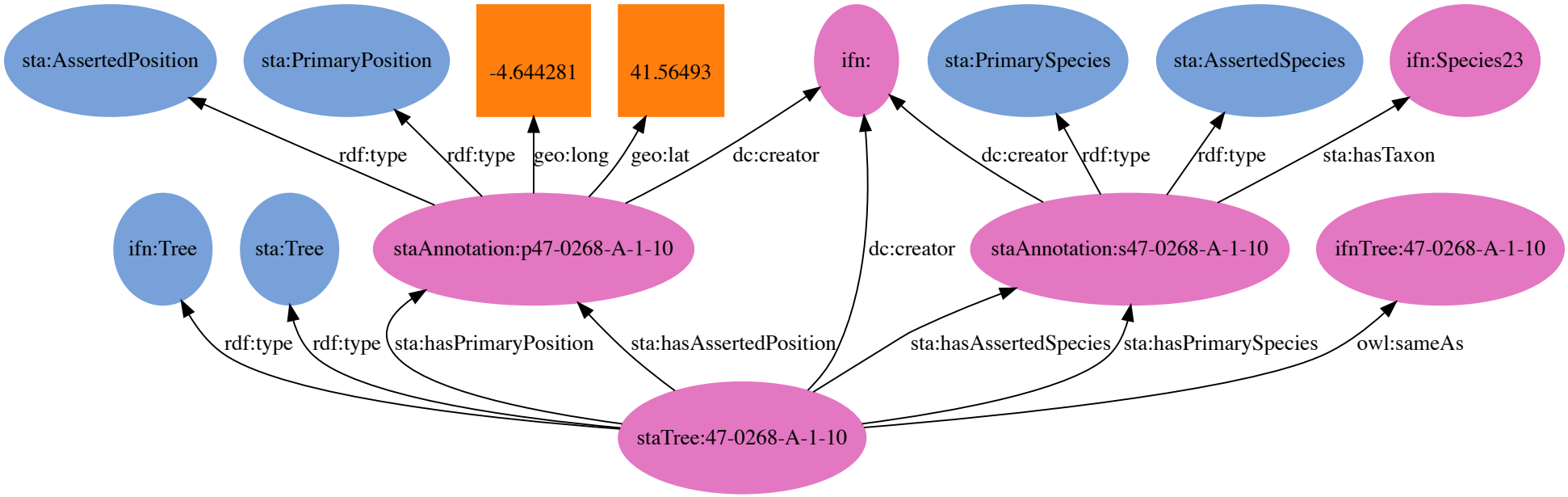


FIGURA 4.12: Representación de triplas de un árbol del IFN y sus anotaciones en la ontología diseñada (STA).

4.5 Diseño de la API REST

El siguiente paso en la etapa de diseño tras haber modelado la ontología y los datos es el diseño de la API REST que se va a exponer. Además de la API, y en estrecha relación con los datos, se deben concretar los IRIs –y los UUIDs (*Universally Unique Identifier*)– que se van a utilizar, ya que no se establecieron completa y firmemente en el apartado 4.2.3. Cabe destacar que el diseño de la API no está sujeto al uso de tecnologías concretas, únicamente al modelo arquitectónico REST.

Previamente a presentar el modelo de recursos, que es la herramienta por antonomasia de diseño de la arquitectura ROA (*Resource Oriented Architecture*) que aplica los principios REST para el desarrollo de servicios web o APIs [43], es importante tratar una cuestión de "compatibilidad" en el diseño de los recursos de la Web Semántica –en concreto, los Datos Enlazados– y REST, ya que sus principios de diseño son diferentes. Esto no quiere decir que sean incompatibles, en realidad las diferencias no son realmente importantes y están más relacionadas con ciertas prácticas o demandas actuales [66]. Los puntos comunes más reseñables entre ambos son el uso de recursos identificables como unidad básica de información, el enlazado de estos, la adaptabilidad y que el diseño de las aplicaciones está impulsado por el dominio (*Model Driven Design*). En cuanto a las diferencias, REST está orientado a las APIs, es decir, los recursos y sus relaciones se identifican y exponen para permitir al cliente recuperar datos y navegar a otros recursos. Se define una API para habilitar operaciones de aplicación y transferencia de estados. RDF y las ontologías, se orientan al modelo, los recursos se identifican para encapsular lo que subyace bajo el modelo de datos. Otra cuestión es que LD distingue los recursos que no son de información (identificadores de conceptos y objetos del mundo real) y de recursos de información (documentos) que describen a los recursos anteriores (los de tipo no información), añadiendo semántica a los IRIs. Por ejemplo, un recurso de no información sería el individuo de tipo `Image` de la ontología y un recurso de información el fichero *JPG* con la imagen propiamente dicha.

Si embargo, como ya se ha comentado, estas disimilitudes no hacen imposible o inadecuada la integración de ambas tecnologías en una aplicación ni el desarrollo de una API REST que exponga datos enlazados. En lo que respecta a las orientaciones API frente a Modelo, los autores [66] dicen que se trata de un aspecto complementario más que una diferencia y señalan la elaboración de un modelo de dominio que separe las preocupaciones de manipulación de los datos (con RDF) y el acceso modificación y publicación de los datos a través de una API (REST), tal y como se está planteando a lo largo de este capítulo. Además, si se pueden utilizar modelos comunes tanto para el diseño de la API (las interacciones de REST con los recursos) como para el modelado de las relaciones con los recursos (el RDF y las ontologías), entonces el foco de la complejidad en cualquier aplicación y en esta en particular subyace en el diseño del dominio [66].

Dentro de este contexto, se identifican los recursos de REST en consonancia con el dominio y la ontología diseñada. Siguiendo los principios de REST se modelarán recursos de tipo entidad y colección, que vienen determinados y se corresponden con las clases definidas en la ontología propuesta y en las ontologías a reutilizar. Aunque típicamente los recursos de tipo colección en REST utilizan IRIs con nombres en plural (nombre de la clase que agrupa un conjunto de recursos entidad), en este caso se utilizarán nombres en singular para poder utilizar los mismos IRIs para identificar los recursos expuestos mediante la API y RDF. Adicionalmente, se evitará el anidado de recursos para evitar tener IRIs excesivamente largos y complejos [2].

4.5.1 Modelo de recursos REST

En la Figura 4.13 se ilustra el modelo de recursos completo de la API REST. Se incluye a mayores un detalle del contenido estático, es decir, de los recursos imagen en formato JPG que se sirven. Para cada recurso se especifica el nombre, el IRI relativo (excepto para el recurso Raíz (*Root*), que es absoluto) y los métodos permitidos así como el formato de los datos que reciben y devuelven.

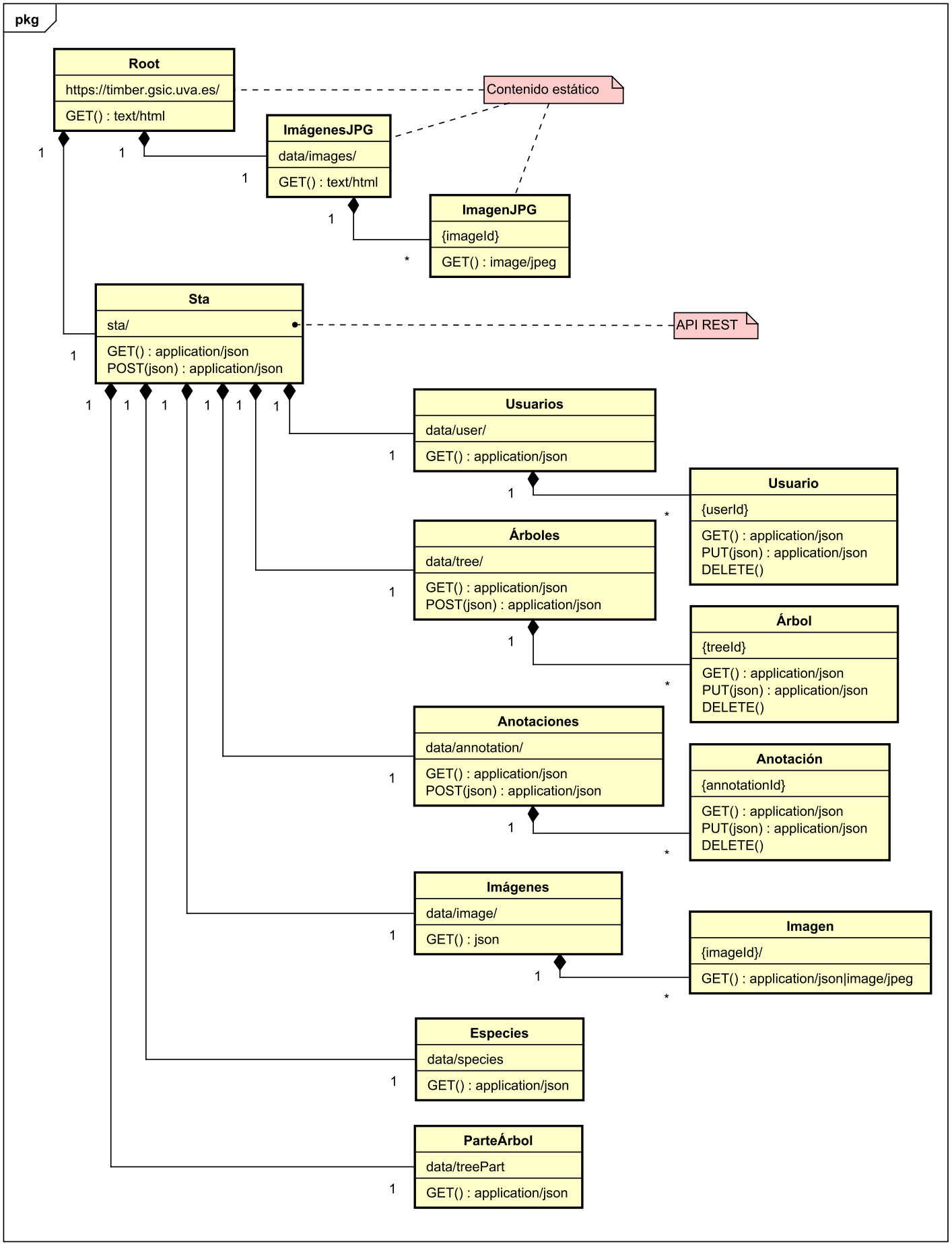


Figura 4.13: Modelo de recursos REST.

4.5.2 Diseño de IRIs de recursos e identificadores

El formato típico del IRI de un recurso es la unión del IRI base, la secuencia de todos los nombres de recursos superiores en la jerarquía de recursos y el nombre del recurso. Si un recurso contiene varios recursos hijos del mismo tipo, el IRI asociado a estos hijos termina con un identificador único. Los formatos elegidos para estos identificadores únicos son los siguientes:

- **Árbol**: `YYYYMMDD-XXXXX`, siendo `YYYY` el año, `MM` el mes, `DD` el día de la creación de árbol y `XXXXX` caracteres aleatorios únicos. Este identificador se denomina `{idTree}` en adelante.
- **Anotación de posición**[1]: `p-{idTree}_XXXXX`, siendo `{idTree}` el identificador del árbol al que pertenece la anotación y `XXXXX` caracteres aleatorios únicos.
- **Anotación de especie**[1]: `s-{idTree}_XXXXX`, siendo `{idTree}` el identificador del árbol al que pertenece la anotación y `XXXXX` caracteres aleatorios únicos.
- **Anotación de imagen**: `i-{idTree}_XXXXX`, siendo `{idTree}` el identificador del árbol al que pertenece la anotación y `XXXXX` caracteres aleatorios únicos.
- **Usuario**: `{idUser}`, siendo el *nick* o nombre de usuario único en el sistema con el que *loggearse*.
- **Imagen**: `{idTree}_XXXXX`, siendo `{idTree}` el identificador del árbol al que pertenece la imagen y `XXXXX` caracteres aleatorios únicos.
- **JPG**: `{idTree}_XXXXX.jpg`, siendo `{idTree}` el identificador del árbol al que pertenece la imagen y `XXXXX` caracteres aleatorios únicos.

Los árboles y anotaciones creados a partir de la importación de los datos del IFN mantienen sus identificadores:

- **Árbol**: `{idTree}`, siendo `{idTree}` el identificador del árbol en el IFN.
- **Anotación de posición**[1]: `p{idTree}`, siendo `{idTree}` el identificador del árbol al que pertenece la anotación.
- **Anotación de especie**[1]: `s{idTree}`, siendo `{idTree}` el identificador del árbol al que pertenece la anotación.

Algunos ejemplos de IRIs completos de recursos se pueden ver en la Tabla 4.3. A su vez estos IRIs hacen referencia a individuos del modelo de datos, es decir, datos concretos.

Adicionalmente los IRIs pueden contener consultas, con el formato *?clave=valor* en su parte final. Por ejemplo, `http://timber.gsic.uva.es/sta/data/tree?creator=demo`, permite recuperar los árboles cuyo creador sea el usuario con *nickname* `demo`. El detalle de las distintas consultas permitidas para cada método de cada recurso se especifica en la documentación del Apéndice B sección *query*.

4.5.3 Documentación de la API: métodos y códigos de estado

Además del modelo de recursos, en el que se especifican los recursos existentes en el *backend* (junto con sus IRIS), los métodos que se exponen y los tipos de datos que requieren/devuelven estos métodos; para la posterior implementación de la API o para su uso –ya sea un desarrollador o un usuario

[1]Válido para cualquier especialización, sea primaria o afirmada.

Elemento	Individuo/Recurso (ejemplo)
Árbol	`http://timber.gsic.uva.es/sta/data/tree/20200621-e30b3`
Anotación de especie	`http://timber.gsic.uva.es/sta/data/annotation/s-20200621-4fea2_d44e2`
Anotación de posición	`http://timber.gsic.uva.es/sta/data/annotation/p-20200621-e30b3_f1d90`
Anotación de imagen	`http://timber.gsic.uva.es/sta/data/annotation/i-20200621-e30b3_8d5e8`
Imagen	`http://timber.gsic.uva.es/sta/data/image/20200621-e30b3-7e211`
Imagen JPG	`http://timber.gsic.uva.es/data/images/20200621-e30b3-7e211.jpg`
Usuario	`https://timber.gsic.uva.es/sta/data/user/cmayo`
Árbol importado IFN	`http://timber.gsic.uva.es/sta/data/tree/47-0003-A-1-1`
Anotación de posición de árbol IFN	`http://timber.gsic.uva.es/sta/data/annotation/p47-0003-A-1-1`
Anotación de especie de árbol IFN	`http://timber.gsic.uva.es/sta/data/annotation/s47-0003-A-1-1`

Tabla 4.3: Ejemplos de IRIs para los recursos de la API.

convencional– es importante conocer los códigos de estado que se envían como respuesta (tanto en caso de éxito como de error). De la misma manera, también se requiere un diseño más exhaustivo que incluya los parámetros a utilizar, errores... En resumen, información más detallada respecto a la API REST para que esta se pueda codificar en la etapa de implementación. A su vez, esta documentación[2] será imprescindible una vez desplegado el sistema para que la API pueda ser utilizada por agentes externos. Se puede consultar en el Apéndice B.

4.6 Conclusiones

El diseño ha establecido cómo va a ser el sistema. Se ha presentado su arquitectura, detallando cada uno de los componentes que conforman el *back-end*. Después, se ha procedido al diseño de la ontología que se propone para la aplicación. Para ello, se ha partido del análisis del modelo de datos del capítulo 3. Se ha explicado la necesidad de crear una ontología, ya que, antes de crear una nueva ontología, se han analizado ontologías existentes y se han explicado detalladamente las que se pueden reutilizar para la aplicación. Los términos del dominio que no se encuentran en ninguna ontología han conformado la ontología creada para la anotación social de árboles.

Una vez se ha establecido el modelo de datos, se han explicado los datos generados por los usuarios e importados del IFN y la forma que tienen, teniendo en cuenta que se representan en las ontologías propuestas. Estos datos son triplas en formato RDF que se guardan en un almacén de triplas, se pueden consultar mediante SPARQL y se ofrecen como LOD.

Respecto al arranque de la aplicación y la ausencia de datos en un primer momento, cabe destacar el problema de *cold-start* típico de las aplicaciones sociales. Este se pretende solventar mediante la importación de datos del IFN en la aplicación y se ha demostrado que las tecnologías de la Web Semántica

[2]Documentación elaborada teniendo en cuenta únicamente la parte de la API REST de la arquitectura, obviando el contenido estático y sabiéndo que la URL raíz de esta API es `http://timber.gsic.uva.es/sta/`. Se incide en que la información presentada está muy resumida: faltan los datos concretos que se devuelven dentro del JSON de respuesta. Además para todos los recursos, se devuelve `405 (MethodNotAllowed)` para los métodos no soportados por cada recurso.

permiten integraciones de datos de forma muy sencilla.

Para terminar, se han tratado cuestiones de compatibilidad entre APIs REST y la Web Semántica y se puede concluir que es posible exponer LOD mediante una API REST realizando un diseño basado en modelos comunes de dominio. Esta API web permite que los datos de la aplicación sean utilizados fácilmente por cualquiera. Por tanto, las contribuciones más importantes han sido respecto al diseño de la ontología propuesta, los datos expuestos como LOD y la API para acceder a estos datos.

Capítulo **5**

Implementación y despliegue del servidor

Tras el diseño de la parte servidora de la aplicación es necesario seleccionar, instalar, configurar e implementar cada uno de los componentes según se requiera. En una primera sección se abordarán las tecnologías elegidas para componente de la arquitectura, así como las justificaciones pertinentes a las decisiones tomadas. En la segunda sección se desarrolla la implementación de solución de la API REST planteada con Node.js, explicando la estructura de directorios y y ficheros del código de esta, su contenido y los principales *frameworks* y módulos utilizados. Por último, con todos los componentes desarrollados, se detalla la puesta en producción del sistema, es decir, se prepara para que esté disponible para los potenciales usuarios mediante la instalación y configuración de los servicios relativos al almacén de triplas OpenLink Virtuoso y el servidor de contenido estático y *proxy* inverso Nginx. Además, se aplica un conjunto de buenas prácticas, algunas de ellas realizadas a nivel de código durante el proceso de codificación de la API, y la configuración del entorno del sistema.

5.1 Tecnologías

La arquitectura del sistema ha sido descrita mediante componentes genéricos, pero es necesario seleccionar y elegir las tecnologías concretas que se van a utilizar para cada uno de ellos como paso previo a la implementación/configuración. Es importante señalar que no existe una tecnología ni un lenguaje de programación óptimos para cualquier situación, sino que estas decisiones vendrán motivadas por la aplicación concreta, así como las condiciones materiales presentes. Un punto común para todas las decisiones tomadas en la elección de las tecnologías de este Trabajo Fin de Máster es que se aboga por herramientas de uso libre y código abierto.

Dentro de los sistemas operativos libres para servidores, existen opciones basadas en Unix como FreeBSD o alguna de las múltiples distribuciones de Linux. Se ha escogido **Ubuntu Server 18.04 LTS**, por ser una versión estable, ampliamente conocida, adecuada para los requisitos a cumplir y que ya se ha utilizado previamente.

Para el servidor web y *proxy* inverso, las alternativas más populares en Linux son Apache o Nginx. Se va a utilizar **Nginx** [22], ya que es adecuado para las dos funciones. Según su propia documentación, es más ligero, requiere hasta un 90 % menos de memoria que otros servidores web como Apache y es mucho más rápido, lo que se traduce en un mayor rendimiento. Permite cachés de contenido estático y dinámico y su configuración es relativamente sencilla. El uso de Nginx como *proxy inverso*, incluye funciones de aceleración web, ya que permite comprimir los datos entrantes y salientes, así como el contenido solicita-

do comúnmente en la memoria caché, lo que acelera el flujo de tráfico entre cliente y servidor. También puede realizar tareas adicionales como encriptación SSL y proporciona un nivel adicional de control y seguridad, ya que intercepta todas las solicitudes dirigidas a los servidores respaldados detrás de él (en este caso a la API y el servicio de *endpoint* del almacén de triplas).

En el caso de la API REST, las opciones también son muy diversas. Se ha optado por una solución con **Node.js** [67] y **Express** [68] por una serie de motivos que se justificarán tras una pequeña explicación previa. Node.js es un entorno de ejecución para JavaScript de código abierto y multiplataforma construido con el motor de JavaScript V8 de Chrome fuera del navegador, orientado a eventos asíncronos y diseñado para crear aplicaciones en red escalables. Es capaz de atender numerosas conexiones simultáneamente con un solo servidor sin introducir la carga de administrar la concurrencia de hilos. Se ejecuta en un solo proceso, sin crear un nuevo hilo para cada petición, es decir, no es un sistema multi-hilo, pero eso no significa que no pueda aprovechar los múltiples núcleos de su entorno. Utiliza devoluciones de llamada o *callbacks* para cada conexión y cuando no tiene trabajo que realizar se queda en estado latente, es decir, en lugar de bloquear el hilo y desperdiciar ciclos de CPU esperando, Node.js reanudará las operaciones cuando vuelva la respuesta. Incluye un bucle de eventos al que se entra después de ejecutar el script de entrada y sale cuando no hay más *callbacks* para realizar. HTTP es un elemento destacado de Node.js, diseñado teniendo en cuenta la transmisión de operaciones con *streaming* y baja latencia. Esto hace que Node.js sea muy adecuado para la base de una librería o un *framework* web. Tiene la ventaja de permitir escribir código JavaScript del lado del servidor –lenguaje del cual se tenían unos conocimientos previos al inicio de este Trabajo Fin de Máster– y dispone de numerosos *frameworks*, herramientas, bibliotecas y módulos construidos por la comunidad, fácilmente gestionables con el gestor de paquetes de Node.js Npm [69]. Una de las bibliotecas más conocidas de Node.js es Express, que permite crear un servidor web o una API REST con Node.js de forma muy sencilla, con un número de líneas de código mucho menor que con otras alternativas como las ofrecidas por otros *frameworks* como Restlet. Por tanto, la elección de estas tecnologías está basada no solo en la simplicidad que presenta la construcción de una API con estas dos herramientas y el conocimiento previo de JavaScript, sino también porque se ajusta a las necesidades del sistema, el que se espera que reciba y responda a un gran número de peticiones de forma asíncrona y con baja latencia. Adicionalmente, cabe señalar que para el intercambio de datos en el sistema y por la red en formato JSON JavaScript es especialmente adecuado (ver requisito RNF-005:).

En el mismo orden de ideas, la caché descrita en la arquitectura del sistema representada en la Figura 4.1, se modelará dentro del servicio web de la API REST como un objeto JavaScript, con una estructura adecuada al contenido que se espera almacenar y que permita la recuperación eficiente de datos de la caché.

Por último, para el almacén de triplas RDF se ha seleccionado OpenLink Virtuoso 7.2.5 Open-Source Edition [31], una plataforma híbrida de almacén de triplas y *middleware* basada en RDBMS de alto rendimiento y escalable para el desarrollo de datos enlazados que soporta SPARQL 1.1 integrado en SQL para consultar datos RDF almacenados en la base de datos de Virtuoso. Es una de las opciones más populares, usada en grandes proyectos de Web Semántica como DBpedia [46], con opción de código abierto, que supera los requisitos mínimos y es más rápida que otras alternativas como el almacén de triplas ofrecido por Oracle [49]. Por todas estas razones será la opción elegida para la parte servidora correspondiente de la aplicación desarrollada en este Trabajo Fin de Máster.

5.2 API REST con Node.js

La API REST se podría considerar como el principal componente del sistema, si bien es cierto que no podría funcionar sin el resto. La capa de aplicación expone la API REST definida en la sección 4.5. Esta se implementa con Node.js y hace uso de la capa de acceso a datos (Virtuoso). Al haber elegido Node.js para su implementación, este componente requiere de programación con JavaScript, es decir, es desarrollado y programado con la ayuda de varias librerías o módulos para satisfacer los requisitos e implementar los casos de uso y la lógica asociada.

El primer paso en la implementación de una API REST con Node.js es la instalación de Node.js en el servidor Linux. La versión utilizada en este trabajo es v12.16.1 ya que es la incluida en los paquetes de la distribución de la versión de Ubuntu instalada. Adicionalmente, se instala el gestor de paquetes de Node.js NPM [69] (versión 6.13.4). Posteriormente, se crea un proyecto de Node.js con `npm init` y se rellenan los valores de configuración que solicita.

5.2.1 Estructura del código

El código completo se puede consultar en [70]. La estructura general de la API REST a nivel de código se puede observar en la Figura 5.1. Se compone de 4 directorios creados de forma manual: **`config`**, contiene los ficheros relacionados con la configuración, IRIs y métodos expuestos para los recursos de la API y códigos de respuesta; **`controllers`**, comprende la lógica de negocio, es decir, la implementación de los métodos que se exponen para cada recurso; **`helpers`**, engloba ficheros con funciones genéricas y relacionadas con consultas de datos; y **`models`**, contiene el modelado de los datos en relación a la caché y a los usuarios. Hay un directorio a mayores, **`node_modules`**, que se crea y se reconstruye automáticamente al instalar algún paquete con NPM y contiene los módulos (librerías JavaScript) necesarias para la API.

Para cada directorio se tienen los siguientes ficheros:

1. **`config`**:
   - *`config.js`*: establece todos los parámetros de configuración de la API REST: puerto donde escucha la aplicación, IRI del *endpoint* SPARQL, grafo por defecto, usuario y contraseña de Virtuoso, tipo de autenticación en Virtuoso, límite y *offset* para las consultas SPARQL, longitud de la parte aleatoria de los UUIDs de los recursos que se crean mediante POST, periodo para la comprobación del estado de la memoria libre disponible, directorio del servidor donde se almacenan las imágenes que suben los usuarios, IRI donde se exponen las imágenes a través del servidor Nginx, fichero que almacena los usuarios y contraseñas de la aplicación, factor de coste para calcular los *hashes* de las contraseñas y límite de la memoria RAM en MB antes de vaciar la caché.
   - *`errorCodes.js`*: define los códigos de error de la API REST y el mensaje que se devuelve para cada uno de ellos.
   - *`httpCodes.js`*: define los códigos de respuesta de la API REST y el mensaje que se devuelve para cada uno de ellos.
   - *`onturis.js`*: contiene los IRIs de las clases y propiedades de las ontologías utilizadas y de los datos de la aplicación.
   - *`queries.js`*: comprende todas las consultas SPARQL que realizará la API sobre el *endpoint*.

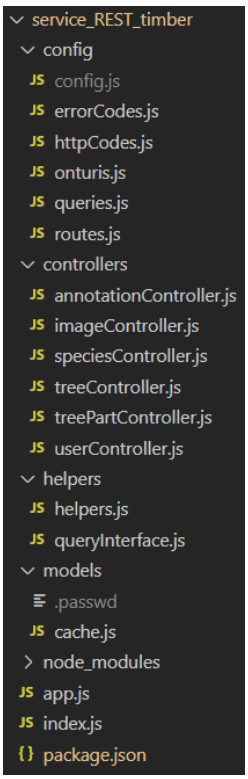


Figura 5.1: Estructura del código de la API REST con Node.js.

- *routes.js*: fichero esencial para la comprensión de la API, ya que, básicamente, es la traducción directa del modelo de recursos a código JavaScript. Incluye el manejo y gestión las peticiones de forma asíncrona y aislada de la lógica de negocio. Al ser funciones asíncronas se puede resolver un número muy elevado de peticiones porque no se bloquea la ejecución a la espera de resolver cada petición, sino que continúa aceptando peticiones y envía la respuesta cuando la promesa que devuelve el método del controlador correspondiente se haya resuelto (o rechazado). Para cada recurso se define la URL correspondiente, los métodos que se exponen y cómo se responde a las peticiones de los clientes. Si la respuesta está cacheada en el servidor a través del módulo `memory-cache` (ver apartado 5.2.2), la respuesta se devuelve directamente. En caso contrario, el controlador se encargará de resolver la petición.

2. `controllers`:

   - *annotationController.js*: controlador de los recursos *Annotations* y *Annotation*. Implementa los métodos de estos recursos y devuelve promesas que son gestionadas por *routes.js* para la elaboración de las respuestas que se envían a los clientes. Los métodos implementados permiten: recuperar las anotaciones de un creador, ver una anotación concreta y crear una nueva anotación (también en el Virtuoso).
   - *imageController.js*: controlador del recurso *Image*. Implementa los métodos que permiten: recuperar una imagen concreta (imagen entendida como individuo de la ontología), subir una imagen JPG al servidor web, decodificar una imagen codificada en *base64* y almacenarla en

un fichero *.jpg*, establecer los metadatos Exif (ancho, alto, fecha y geolocalización) de una imagen si los tuviera y crear un nuevo recurso imagen en el sistema (también en Virtuoso).

- *speciesController.js*: controlador del recurso *Species*. Implementa los métodos que permiten recuperar la jerarquía (taxonomía) completa de especies cargada en el Virtuoso.
- *treeController.js*: controlador de los recursos *Trees* y *Tree*. Implementa los métodos de estos recursos y devuelve promesas que son gestionadas por *routes.js* para la elaboración de las respuestas que se envían a los clientes. Los métodos implementados permiten: recuperar todos los árboles del sistema, los creados por un usuario concreto, los localizados en un área concreta o los de una especie determinada, ver un árbol concreto y dar de alta un árbol en el sistema. (también en el Virtuoso).
- *treePartController.js*: controlador del recurso *TreePart*. Implementa el método que permite recuperar las partes de un árbol que se pueden seleccionar en una foto (subclases de la clase *TreePartPhoto* de la ontología STA) cargadas en el Virtuoso.
- *userController.js*: controlador de los recursos *Users* y *User*. Implementa los métodos de estos recursos y devuelve promesas que son gestionadas por *routes.js* para la elaboración de las respuestas que se envían a los clientes. Los métodos implementados permiten: recuperar todos los usuarios registrados en el sistema, ver los datos de un usuario concreto, crear un nuevo usuario en el sistema (con los datos "públicos" en el Virtuoso y el *hash* de la contraseña en el fichero *.passwd*), iniciar sesión en el sistema y comprobar el login y contraseña de un usuario existente.

3. `helpers`:

   - *helpers.js*: abarca varias funciones genéricas para terminar la ejecución del servidor que ejecuta la API REST (admite varias señales de salida), generar los UUIDs de los recursos que se crean con métodos POST, obtener la fecha del momento de creación de un recurso en formato ISO, convertir coordenadas de formato grados, minutos, segundos a grados decimales y obtener las coordenadas GPS de los metadatos Exif de una imagen.
   - *queryInterface.js*: contiene las funciones que permiten realizar consultas SPARQL y recuperar los datos que se requieran del Virtuoso. Adicionalmente, incluye una función recuperar todos los datos de un individuo en concreto del Virtuoso y cachear y *parsear* los resultados adecuadamente y otra que establece un vector con consultas SPARQL independientes que pueden realizarse simultáneamente. Un ejemplo de esta última funcionalidad se da en el caso de los árboles, para recuperar los datos básicos que se muestran al listar los árboles del sistema, se pueden lanzar distintas funciones asíncronas (promesas) para consultar los datos de cada propiedad de todos árbol (por ejemplo, la posición, especie o creador). Una vez se hayan resuelto todas las promesas se tienen todos los datos en memoria y se pueden formatear adecuadamente, cachearlos y formar la respuesta.

4. `models`:

   - *cache.js*: define varios objetos (`trees`, `annotations`, `images`, `species`, `users`) para almacenar en memoria el contenido de las consultas que se vayan realizando al Virtuoso. Cuando se reciba una petición de consulta sobre un recurso que ya ha sido guardado en la caché no se repetirá la consulta al Virtuoso, reduciendo el tiempo de respuesta y las transacciones con

el almacén de triplas (es relativamente lenta esta comunicación, en comparación con recuperar datos en memoria). Cuando se crea un nuevo recurso en el sistema también se almacena en la caché. Estos objetos tendrán estructura *clave-valor*, siendo la clave el IRI del recurso (coincide con el identificador almacenado en Virtuoso) y el valor las distintas propiedades y valores de cada una de ellas, de forma que recuperar los datos es muy eficiente. Un ejemplo de la estructura de la caché para los árboles es el siguiente:

```
trees = {
  "http://timber.gsic.uva.es/sta/data/tree/20200615-f859c" : {
      "creator" : http://timber.gsic.uva.es/sta/data/user/jimena22,
      "lat" : 41.654681599999996,
      "long" : -4.6956544,
      "species" : http://crossforest.eu/ifn/ontology/Species235,
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/ontology/Tree
      },
      "http://purl.org/dc/elements/1.1/creator" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/data/user/jimena22
      },
      "http://timber.gsic.uva.es/sta/ontology/hasImageAnnotation" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/data/annotation/i-20200615-f859c_806d1
      },
      "http://timber.gsic.uva.es/sta/ontology/hasPositionAnnotation" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/data/annotation/p-20200615-f859c_b65f2
      },
      "http://timber.gsic.uva.es/sta/ontology/hasPrimaryPosition" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/data/annotation/p-20200615-f859c_2a9c0
      },
      "http://timber.gsic.uva.es/sta/ontology/hasPrimarySpecies" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/data/annotation/s-20200615-f859c_2893f
      },
      "http://timber.gsic.uva.es/sta/ontology/hasSpeciesAnnotation" : {
        "type" : uri,
        "value" : http://timber.gsic.uva.es/sta/data/annotation/s-20200615-f859c_1b791
      },
      "http://purl.org/dc/elements/1.1/created" : {
        "type" : typed-literal,
        "datatype" : http://www.w3.org/2001/XMLSchema#date,
        "value" : 2020-06-15T14:10:15.999
      }
  },
  "http://timber.gsic.uva.es/sta/data/tree/47-0226-A-3E-15": {
      ...
  },
  ...
}
```

Además de la propia caché modelada, se definen e implementan varios métodos asociados con

el cachéo de datos, la limpieza de la caché y la obtención de la memoria RAM disponible del servidor.

Por otro lado, incluye la creación, configuración y los métodos de recuperación y almacenamiento de otra caché procedente de un módulo (`memory-cache`) y que actúa como *middleware* (más detalles en la sección 5.2.2).

- *`.passwd`*: fichero que almacena los nombres de usuarios registrados en el sistema y los *hashes* de las contraseñas asociados a cada uno de ellos.

Asimismo, externos a los directorios, hay 3 ficheros:

- *`app.js`*: es el fichero principal (*main*) del proyecto Node.js, en el cual se define la aplicación (entendida como la API REST), los métodos relacionados con la inicialización y arranque de la API (inicialización de la API, configuración de *middlewares* utilizados y arranque de la API en el puerto especificado) y las llamadas a funciones como la conexión con el *endpoint* SPARQL. También llama cada cierto intervalo de tiempo a una función que comprueba la memoria RAM disponible en el servidor físico para vaciar la caché si fuera necesario.
- *`index.js`*: establece la conexión con el *endpoint* SPARQL y su configuración inicial. Devuelve un objeto que representa al cliente SPARQL que se utilizará para realizar todas las consultas y comunicaciones con el punto SPARQL.
- *`package.json`*: fichero que se añade al crear el proyecto Node.js y facilita que este sea fácil de manejar e instalar por terceros, ya que, aparte de incluir información como el nombre, el autor o la versión, lista todas las dependencias y las versiones de cada paquete que se pueden utilizar e indica el fichero principal, entre otras muchas opciones que se pueden añadir o personalizar.

5.2.2 Principales frameworks y módulos utilizados

Típicamente, en la implementación de cualquier sistema se utilizan una serie de *frameworks* y librerías (o módulos, en el caso de JavaScript) para facilitar la tarea al programador.

5.2.2.1 Express

En este sistema desarrollado con Node.js el principal *framework* empleado es Express [68]. Es una capa delgada que proporciona métodos HTTP y *middleware* característicos de aplicaciones web que permiten la creación de una API sólida de forma rápida y sencilla. Básicamente, la programación de una API con estas tecnologías consiste en definir una aplicación y el enrutamiento (*routing*) de las peticiones (definir los IRIs de los recursos y los métodos HTTP). Un ejemplo sencillo en código de estas funcionalidades puede ser el que se muestra (para ver el código completo se pueden consultar los ficheros *app.js* y *routes.js* del código de la aplicación):

```
app = express();
app.listen(config.port, () => {
        console.log('%s: Node server started on %d ...',
        Date(Date.now()), config.port);
});
app.get('/', function (req, res) {
  res.send('hello world')
});
```

Un particularidad de Express es que permite el uso *middlewares* [68, 69] –funciones que tienen acceso al objeto de la petición, al objeto de la respuesta y a la siguiente función del *middleware* en el ciclo de petición-respuesta de la aplicación– que se invocan por la capa de enrutamiento de Express antes del gestor de peticiones final, siendo, por tanto, capas intermedias que añaden funcionalidad a distintos niveles. En este sistema se emplean principalmente los *middlewares*: `cors`, permite habilitar CORS[1] (*Cross Origin Resource Sharing*) para una o varias rutas; `compression`, para comprimir los cuerpos de respuesta de las peticiones, `body-parser`, parsea el cuerpo de petición y, con el método `.json()`, solo admite peticiones HTTP con cabeceras `Content-Type` de tipo JSON; y `memory-cache`, permite construir una caché en el servidor utilizando las URLs de las peticiones como clave.

Gracias al módulo `body-parser` se puede limitar el tamaño máximo del cuerpo de la petición. Esto es especialmente útil para el caso de las imágenes subidas a la plataforma, ya que permite limitar el tamaño de estas, aunque, también, como medida de seguridad frente a posibles ataques (como los de inyección de código). Con estos objetivos se permitirán peticiones de hasta 15MB. Dado que las imágenes se codifican en *base64* aproximadamente podrán tener un tamaño de entorno a 10MB, teniendo en cuenta la relación con los bytes y el resto del cuerpo de la petición: $15MB \cdot \frac{6}{8} = 11{,}25MB$.

Respecto al módulo de la caché *middleware*, `memory-cache`, se configura para los métodos de tipo GET. Almacena en un objeto los resultados de las peticiones y utiliza las URLs de estas como clave, de forma que es muy rápido recuperar los resultados. Ahora bien, la frecuencia de refresco de esta caché tiene que ser elevada, ya que i se crea un nuevo recurso como un árbol o anotación la caché no conocerá su existencia. Por ello, se configura una validez de 30 segundos para el caso de los árboles y anotaciones y de 30 minutos para contenido "estático" como las especies y las partes de los árboles a etiquetar en una foto. Debido a esta capa intermedia no se tiene un sistema con respuesta en tiempo real (aunque podría considerarse *pseudo* real, ya que los tiempos de la caché son cortos y tampoco es crítico para esta aplicación tener los datos actualizados en tiempo real), pero es capaz de resolver las peticiones de consulta sobre los mismos recursos en un tiempo más reducido y atender una cantidad mayor de peticiones.

5.2.2.2 uuidv4

Como su nombre indica, es un módulo que permite crear UUIDs versión 4 (aleatorios) [69]. Se utiliza para la generación de los identificadores únicos de los recursos. La implementación de los UUIDs empleados por la aplicación es la siguiente:

```
const { uuid } = require('uuidv4');

function generateId(){
    var id ="";
    id = uuid();
    id=id.substring(id.length-config.lenghtId);
    var today = new Date();
    var dd = today.getDate();
    var mm = today.getMonth()+1; //January is 0!
    mm = (mm <10)? "0" + mm : mm;
    dd = (dd <10)? "0" + dd : dd;
    var yyyy = today.getFullYear();
```

[1] Útil en el caso de integración con un *front-end* alojado en otra máquina

```
    var full_id = yyyy.toString() + mm.toString() + dd.toString() + "-" + id;
    return {"full_id": full_id, "id": id};
}
```

5.2.2.3 Bcrypt

Librería para crear *hash* de contraseñas [69]. Se utiliza para generar los *hashes* de las contraseñas de los usuarios que se registran en el sistema y para autorizar a los usuarios, comparando la contraseña con el *hash* almacenado en el fichero de usuarios.

```
const bcrypt = require('bcrypt');

bcrypt.hash(password, config.saltRounds).then((hashPass) => {
    //Función que se ejecuta en caso de éxito
    }).catch((err) => {
        //Función que se ejecuta en caso de error
})
...
bcrypt.compare(password, hash, function (err, result) {
    if (result == true)
        //Coindice contraseña y hash
    else
        //No hay coincidencia
});
```

5.2.2.4 Virtuoso SPARQL Client

Es un cliente HTTP para un *endpoint* SPARQL en Node.js [69]. Implementa varios métodos, siendo esencial para esta aplicación la realización de consultas de forma sencilla al *endpoint* SPARQL. Las consultas se realizan mediante el método `query()` de una instancia de un cliente (`Client`).

```
const {Client} = require('virtuoso-sparql-client');

//Inicialización del cliente SPARQL
var sparqlClient = {};
var auth = config.authType + " " + btoa(config.user + ':' + config.pwd);
sparqlClient = new Client(config.endpoint, auth);
sparqlClient.setDefaultFormat("application/json");
sparqlClient.setDefaultGraph();
...
//Ejecución de una consulta SPARQL definida en el parámetro query correctamente formada
sparqlClient.query(query);
```

Adicionalmente, para poder insertar datos en el Virtuoso mediante consultas de inserción (*INSERT*) a través de la ruta `/sparql-auth` (ver sección 5.3.1.1), es necesario extender[2] el módulo permitiendo la autenticación HTTP básica. Se modifica el constructor de `Client` y se añade un parámetro `auth` –es un objeto que contiene el tipo de autenticación y el usuario y contraseña de Virtuoso (con permisos `SPARQL_UPDATE`) codificados correctamente en *base64*– en el fichero *client.js*. Se añade una cabecera HTTP `Authorization` cuyo valor es el objeto `auth` en el objeto de opciones del método `query()`.

[2]Es posible añadir nuevas funcionalidades ya que está publicado bajo licencia MIT.

```
module.exports = class Client {
  constructor(endpoint, auth){
    ...
    this.auth = auth;
  }
  ...
  _createQueryOptions(query, echo){
    ...
    opts.headers = {
    ...
    'authorization': this.auth
    };
    ...
  }
  ...
}
```

5.2.2.5 btoa

Convierte datos binarios a ASCII codificado en *base64*. Se emplea para generar las cabeceras HTTP de autorización (`Authorization`). Un ejemplo de esta cabecera se ilustra a continuación. Además, se puede ver su uso *in situ* en el código en el ejemplo anterior de inicialización del cliente SPARQL.

```
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

5.3 Despliegue en producción

Para desplegar el *software* en producción y que esté disponible para su uso es necesario instalar y configurar una serie de componentes. Se sigue un conjunto de buenas prácticas y configuraciones para aumentar el rendimiento, fiabilidad y seguridad del sistema. El *back-end* se aloja en una máquina, que funcionará como servidor, se le ha asignado una dirección IP (`10.0.104.248`) de la red privada del GSIC/EMIC [11] y un nombre de dominio en el servidor DNS del grupo: `timber.gsic.uva.es`[3], en consonancia con los IRIs diseñados.

5.3.1 Servicios

A partir de la arquitectura diseñada y las tecnologías seleccionadas, los principales servicios a instalar y configurar para la puesta en marcha del servidor de la aplicación web son los relacionados con el almacén de triplas y *endpoint* SPARQL (OpenLink Virtuoso), el servidor web y *proxy* inverso (Nginx).

5.3.1.1 OpenLink Virtuoso

Inicialmente, la instalación del almacén de triplas Virtuoso se realizó a través de un contenedor Docker con una imagen Virtuoso ya que el proceso es mucho más sencillo que una instalación convencional en Linux [71]. Sin embargo, se trata de una imagen no oficial y se detectaron múltiples errores

[3]El nombre procede de la palabra en inglés "madera" (*timber*), como un guiño a la temática forestal y *tinder*, en relación con la parte social.

en la carga y consulta de datos por lo que está opción fue desestimada. En consecuencia, se procedió a la instalación convencional de la versión OpenLink Virtuoso OpenSource Edition 7.2.5 en Linux 64 bits.

La instalación de este *software* no es inmediata, requiere la instalación de las dependencias oportunas, la compilación manual del programa escrito en C y la ejecución de los test que indica la documentación [72] para comprobar que el proceso ha finalizado con éxito. A mayores, se puede optimizar su configuración en función de los recursos de la máquina donde se ha instalado. Las notas seguidas para la instalación y configuración se encuentran en el Apéndice C.

Una vez instalado el almacén de triplas y establecido el acceso al *endpoint* SPARQL, se cargan los ficheros con la ontología y los datos iniciales con los comandos adecuados para realizar esta tarea (ver Apéndice C). Los ficheros cargados son los siguientes:

- ***ontology.ttl***: fichero con la ontología STA diseñada incluida en el Apéndice A.
- ***VLL_trees_ifn.ttl***: contiene los árboles de la provincia de Valladolid existentes en el IFN exportados del conjunto de datos semánticos del IFN mediante la consulta y el procedimiento descrito en la sección 4.4.

Además, se requieren los ficheros que modelan la taxonomía de especies del Cross-Forest [12] en RDF:

- ***clases_onto_crossforest.ttl***: contiene las clases (coníferas y frondosas) de la taxonomía de especies del Cross-Forest.
- ***clases_crossforest_dbpedia.ttl***: identifica y enlaza con `owl:sameAs` las clases anteriores con recursos de la DBPedia que hacen referencia a los mismos individuos pero con distinto IRI.
- ***familias_onto_crossforest.ttl***: define las familias –siguiente categoría taxonómica más específica– de cada clase de la taxonomía de especies forestales .
- ***familias_crossforest_dbpedia.ttl***: identifica y enlaza las clases anteriores que hacen referencia a familias con recursos de la DBPedia que hacen referencia a los mismos individuos pero con distinto IRI.
- ***generos_onto_crossforest.ttl***: contiene los géneros –categoría taxonómica que agrupa a las especies relacionadas entre sí– y los define como subclases con axiomas RDFS de las distintas familias.
- ***generos_crossforest_dbpedia.ttl***: identifica y enlaza las clases definidas anteriormente para los géneros con recursos de la DBPedia que hacen referencia a los mismos individuos pero con distinto IRI.
- ***especies_onto_crossforest.ttl***: contiene las categorías básicas de especies definidas como subclases de los distintos géneros. Es el último escalón de la taxonomía de especies forestales del Cross-Forest.
- ***especies_crossforest_dbpedia.ttl***: identifica y enlaza las clases definidas anteriormente para las especies con recursos de la DBPedia que hacen referencia a los mismos individuos pero con distinto IRI.
- ***names_crossforest.ttl***: define los nombres vulgares de las especies en inglés.

- ***nombres_crossforest.ttl***: define los nombres vulgares de las especies en castellano.

En este punto ya es posible realizar consultas SPARQL en el *endpoint* por defecto de Virtuoso: `http://timber.gsic.uva.es:8890/sparql`. Esto sería suficiente si el objetivo fuese únicamente la consulta de datos pero, como posteriormente se insertarán nuevas triplas, es necesario realizar algunos pasos adicionales en la configuración.

Con el propósito de satisfacer los requisitos propuestos, cualquier persona puede consultar los datos (mediante el uso de la API REST o directamente en el SPARQL) pero solo los usuarios registrados pueden dar crear, modificar o eliminar información (únicamente a través de la API REST). Para insertar datos RDF en Virtuoso hay varios métodos según su documentación [31]. Particularmente, esta aplicación creará nuevas triplas mediante peticiones SPARQL [21] de tipo inserción (*INSERT*) a través del *endpoint*. Para ello, en Virtuoso se exponen dos interfaces o rutas: `/sparql`, para consultas; y `/sparql-auth`, que, a mayores, permite actualizaciones si se dispone de un usuario con privilegios (es un usuario de Virtuoso, no confundir con los usuarios del sistema o de la API). Para esta segunda ruta (*path*), se configura la autenticación básica (*Basic*) como método de autenticación, a través de la interfaz web de usuario de Virtuoso llamada Conductor. También se crea un usuario con privilegios de actualización (`SPARQL_UPDATE`), que será utilizado internamente por la API REST para las creaciones de datos RDF en el almacén de triplas. La idea es que la API REST, con rol de *proxy* entre clientes y el SPARQL, traduzca las peticiones de los usuarios de la aplicación de forma transparente a SPARQL y realice peticiones HTTP a este *endpoint* mediante la ruta relativa `/sparql-auth`, empleando autenticación básica HTTP con el usuario con permisos `SPARQL_UPDATE` creado anteriormente para realizar modificaciones. La ruta absoluta del *endpoint* con el que se comunicará la API independientemente de la operación a realizar es `http://timber.gsic.uva.es:8890/sparql-auth`. Cabe destacar que, previamente, la API tendrá que verificar si el usuario está registrado en la aplicación (en este caso es el usuario de la aplicación) antes de realizar ninguna petición que requiera estar registrado. Esta explicación se esquematiza en la Figura 5.2.

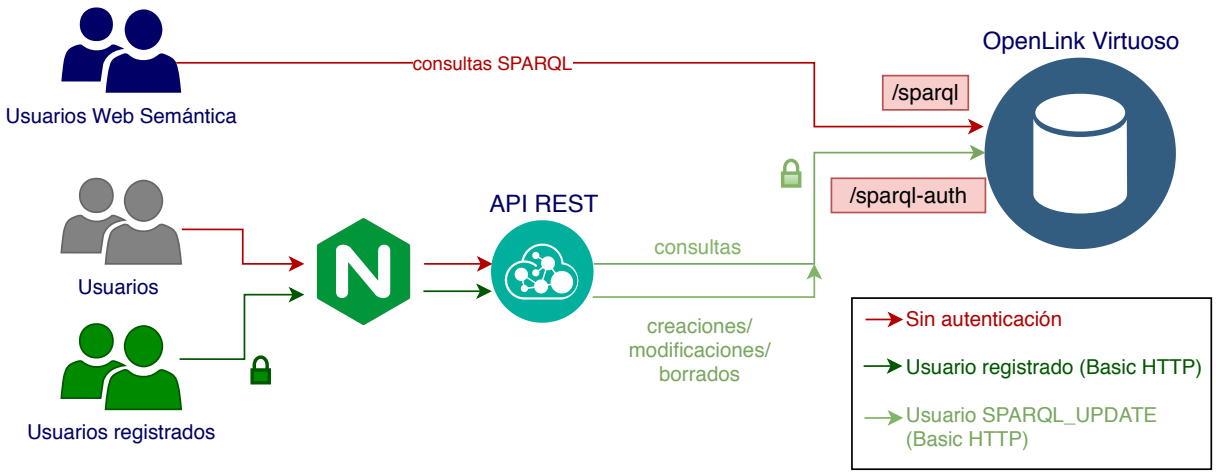


Figura 5.2: Esquema de accesos al *endpoint* SPARQL por distintos tipos de usuarios.

5.3.1.2 Nginx

El servidor web y *proxy* inverso Nginx se va a ejecutar en un contenedor Docker a partir de una imagen con la versión 1.17.10 [30]. Sus funciones básicas son servir las imágenes de la aplicación (contenido estático) y recibir las peticiones de los clientes y reenviarlas a los distintos servidores (tendrá otras funciones más avanzadas que se detallarán en la sección 5.3).

Inicialmente se descarga la imagen del repositorio Docker Hub y se crea un fichero de configuración YAML (*Dockerfile*) para el servicio Nginx que permite ejecutar el contenedor Docker con la herramienta Compose. En este fichero se configura la imagen utilizada, el nombre del contenedor, los volúmenes, los puertos y la opción de que el contenedor se reinicie siempre en caso de que se detenga, falle... [23, 30]

```
nginx:
    image: nginx:latest
    container_name: mynginx
    volumes:
    - /home/ubuntu/nginx/etc/nginx:/etc/nginx:ro
    - /home/ubuntu/nginx/html:/usr/share/nginx/html:ro
    ports:
    - 80:80
    - 443:443
    restart: always
```

De acuerdo a la configuración anterior, los ficheros de configuración de Nginx estarán en las rutas especificadas en el primer volumen y el contenido estático del servidor web en las correspondientes al segundo volumen (a la izquierda del símbolo ":" se especifica la ruta local y a la derecha la ruta dentro del contenedor).

Es necesario configurar un servidor virtual escuchando en el puerto 80 para procesar las peticiones HTTP que se reciban en la IP del servidor o su nombre de dominio. Se establecen varios contextos `location` para definir cómo procesar distintos subconjuntos de IRIs: las asociadas a la API REST (`/sta`), el *endpoint* SPARQL (`sparql` y `sparql-auth`) y las imágenes (`/data/images/`). En los dos primeros casos se utiliza la directiva `proxy_pass` para redirigir el tráfico de esos IRIs al puerto correspondiente. Por ejemplo, la API REST Node.js estará escuchando en el puerto 8888 y el SPARQL en el puerto 8890. Para las imágenes se habilita el listado del contenido dentro del directorio que las contiene [22].

```
server {
    listen   80;
    server_name  timber.gsic.uva.es;
    root /usr/share/nginx/html;

    # Servicio web Nodejs => Backend
    location /sta {
        proxy_pass http://timber.gsic.uva.es:8888;
    }

    # Endpoint SPARQL
    location /sparql {
        proxy_pass http://timber.gsic.uva.es:8890;
    }
```

```
    # Contenido estático: imágenes
    location /data/images/ {
        autoindex on;
    }
}
```

Tras modificar los ficheros de configuración es imprescindible recargar el servidor Nginx del contenedor Docker para que apliquen los cambios.

5.3.2 Buenas prácticas para la API REST

En relación a la API REST implementada con Node.js y Express, la documentación [68] establece recomendaciones a nivel de código y de entorno para producción. A nivel de código se refiere al uso de compresión gzip, utilización de funciones asíncronas y manejo correcto de excepciones (realizado a través de promesas para este sistema). Estas recomendaciones han sido aplicadas durante el proceso de implementación. En el entorno del sistema para mejorar el rendimiento, se han configurado, tras el desarrollo de la API, varios aspectos como: la variable de entorno `NODE_ENV` a "*production*" (mejora hasta tres veces el rendimiento de la aplicación según la documentación), un servicio de `systemd` para reiniciar automáticamente la aplicación si se detiene por cualquier motivo o para que se inicie automáticamente si se reinicia la máquina que lo aloja y el uso de un servidor *proxy* inverso como Nginx para manejar las páginas de errores, la compresión, el almacenamiento en memoria caché y el servicio de archivos.

Además de configurar un servicio de `systemd` para la API REST, también interesa que el Virtuoso se inicie automáticamente si se detiene el proceso o al enceder/reiniciar el servidor. La configuración de los servicios mediante sus ficheros *unit*, el asociado a la API y al Virtuoso, se incluye en el Apéndice D.

El servidor web y *proxy* inverso Nginx se ha configurado con opciones más avanzadas (siguiendo la documentación propia [22] y las buenas prácticas de Express [68]) a las presentadas en el apartado 5.3.1.2 para ofrecer un mayor rendimiento añadiendo: almacenamiento en memoria caché, compresión *gzip* y encriptación SSL para ofrecer sitios disponibles a través de conexiones seguras HTTPS. El fichero de configuración completo se incluye en el Apéndice E.

5.4 Conclusiones

A lo largo de esta capítulo se ha expuesto la implementación y puesta en marcha del servidor de la aplicación de anotación soccio-semántica de árboles, es decir, se ha materializado el *software* que se había diseñado previamente. Antes de comenzar la programación o configuración propiamente, ha sido necesario elegir las tecnologías concretas utilizadas. Estas decisiones se han basado en criterios objetivos en función de los requisitos del sistema y las condiciones específicas. Este proceso ha tenido como resultado la generación de código JavaScript para la API REST y de distintos ficheros de configuración para los servicios de Virtuoso, Nginx o el sistema operativo Linux. A posteriori se puede concluir que estas decisiones han sido acertadas porque la experiencia ha sido positiva.

Hay que señalar que no todas las funcionalidades y requisitos expuestos en los capítulos correspondientes han sido implementados. Es un hecho frecuente en los desarrollos de *software* limitados en tiempo. En particular, la funcionalidad asociada a la selección y anotación de partes concretas de imágenes no se

ha implementado. Las anotaciones que se consideran válidas en un momento dado, son las anotaciones más recientes. No existen categorías de usuarios, por lo que no habrá anotaciones afirmadas más allá de las importadas del IFN. A pesa de estas carencias, el prototipo resultante del proceso iterativo es bastante completo y funcional.

Capítulo **6**

Validación y evaluación

Una vez explicados los métodos utilizados en este Trabajo Fin de Máster, en este capítulo se exponen los resultados obtenidos de la evaluación del sistema, tanto a nivel de validación funcional mediante la realización de distintos casos de prueba como a nivel de rendimiento, centrándose en las medidas cuantitativas de los tiempos de respuesta de las peticiones desde la perspectiva de la API REST fundamentalmente. A partir de estas métricas, obtenidas mediante experimentos sobre el sistema real con pruebas de carga automáticas con cargas sintéticas, se realiza un pequeño estudio y comparativa del rendimiento del sistema debido a las cachés y la influencia del número de usuarios concurrentes. Finalmente, se presenta un ejemplo de integración de la parte servidora del sistema con una interfaz gráfica de tipo aplicación web.

6.1 Metodología de evaluación del sistema

Las pruebas son un aspecto esencial en el ciclo de vida del desarrollo *software* y se definen como procesos sistemáticos que permitan probarlo antes de su uso real [73, cap. 8]. Con el objetivo de evaluar el sistema desarrollado (entendido como la parte servidora o *back-end*) se pueden realizar varios tipos de pruebas a nivel funcional o no funcional y que se pueden categorizar según [57, cap. 8] en: unitarias, de integración, automáticas, del sistema o de aceptación, entre otras.

En este capítulo se presentan pruebas que evalúan o se ejecutan principalmente sobre la API REST, ya que es el componente *software* implementado en este trabajo. Sin embargo, cabe señalar que, especialmente el rendimiento, está condicionado por la configuración de otros componentes, en concreto, el Nginx y el Virtuoso y por la carga de la red y la nube de cómputo donde se aloja la máquina virtual.

Las pruebas que se presentan en este capítulo son de dos grandes tipos, se enmarcan en las categorías de pruebas automáticas y del sistema y se clasifican como: funcionales, garantizan que se cumplen los requisitos funcionales y los casos de uso propuestos en la etapa de análisis; y métricas, cuantifican el rendimiento de la aplicación en conjunto (relacionadas con los requisitos no funcionales) y permiten ofrecer garantías sobre el servicio que se va a ofrecer a los usuarios finales. Otras categorías de pruebas como la validación de la correcta integración entre distintos componentes, entre el almacén de triplas y la API REST o el servidor Nginx, han sido realizadas durante toda la etapa de pruebas pero no se presentan de manera explícita, con el supuesto de que la evaluación del sistema en conjunto implica la correcta integración y funcionamiento de las piezas que conforman el sistema.

Las validación y evaluación del sistema se llevará a cabo principalmente mediante pruebas automatizadas. En el caso de las pruebas funcionales se emplean básicamente varios *scripts de shell* de Linux que utilizan la herramienta cURL [74] para invocar a los métodos de la API REST codificados específicamente para el sistema, aunque también se pueden certificar las funcionalidades de forma manual mediante un navegador o un cliente REST. Por otro lado, las pruebas de rendimiento se realizan mediante test diseñados en JavaScript que invocan a la librería `loadtest` disponible en NPM [69].

6.2 Pruebas funcionales

Estas pruebas se realizan para evaluar si un componente o sistema satisface los requisitos o especificaciones funcionales. Para cubrir todos los escenarios se diseñan una serie de casos de prueba teniendo en cuenta los requisitos funcionales especificados en el apartado 3.3.1. Si el comportamiento del sistema observado es el esperado, se trata como una condición de éxito. En caso contrario, da lugar a una condición de fallo [73, cap 2.].

Para procurar evaluar correctamente el comportamiento del sistema, no solo en el flujo básico sino también frente a fallos, se diseñan casuísticas que produzcan, además de situaciones convencionales, flujos alternativos y desaten situaciones tanto deseables como no deseables. De esta forma se comprueba el funcionamiento de la aplicación y el tratamiento de los errores por parte de la API implementada.

6.2.1 Casos de prueba

Los casos de prueba son una de las herramientas que existen para la evaluación funcional de un sistema *software* y se diseñan sobre la base de la funcionalidad y en relación con los casos de uso, ignorando la estructura interna del programa (pruebas de caja negra) [73, cap. 6, 7]. Típicamente, se presentan en formato de tipo tabla.

En concreto, los casos de prueba diseñados para la aplicación desarrollada se detallan en las Tablas 6.1 y 6.2. Cada uno de ellos tiene: un identificador único, un nombre, los prerrequisitos necesarios para poder realizar la prueba concreta, una descripción del comportamiento esperado, el usuario que lo ha realizado, con qué herramienta se ha probado, fecha de la prueba, resultado y comentarios adicionales. Las pruebas asociadas a los flujos básicos se muestran en la Tabla 6.1. Para este trabajo los flujos alternativos posibles de cada escenario se muestran en Tabla 6.2 y se identifican como un nuevo caso de prueba, con el mismo nombre que el caso base e identificador, pero con la diferencia que presentan entre paréntesis un detalle asociado y un subidentificador. Todos los casos de prueba presentados versan sobre los casos de uso que se han implementado (no se incluyen todos los indicados en la sección 3.4), si bien, algunos casos de uso "complejos" se desdoblan en varios casos de prueba para comprobar todas las restricciones (por ejemplo, el caso de "Ver árboles", **CU-007:**). La correcta lectura de los casos de prueba debe complementarse con los casos de uso de la sección 3.4 y puede apoyarse en la documentación de la API REST presentada en el Apéndice B. Es deseable que todos finalicen éxitosamente, aunque en caso contrario se pueden detectar las deficiencias para la siguiente iteración en el proceso de desarrollo *software*.

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado/Comentarios
B-T001	Ver árboles del sistema	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder consultar los árboles del sistema. El resultado será hasta 1000 árboles en formato json con su información básica: creador, latitud, longitud y especie (si la tiene) y un enlace a la siguiente página (si hay más de 1000 árboles encontrados).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo `https://timber.gsic.uva.es/sta/data/tree/`
B-T002	Ver árboles de una zona	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer las coordenadas del área que se quiere consultar.	El usuario tiene que poder consultar los árboles en una zona delimitada por dos latitudes y dos longitudes cualesquiera. El resultado será los árboles de esa zona en formato json con su información básica.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree?lat0=41.66&long0=-4.72&lat1=41.6606&long1=-4.7239`
B-T003	Ver árboles de una especie	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer la especie que se quiere consultar.	El usuario tiene que poder consultar los árboles de una especie concreta. El resultado será hasta 1000 árboles en formato json con su información básica: creador, latitud, longitud y especie y un enlace a la siguiente página (si hay más de 1000 árboles encontrados).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree?specie=Specie20`
B-T004	Ver árboles creados por un usuario	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el nombre de usuario del creador para el que se quiere consultar.	El usuario tiene que poder consultar los árboles de un creador concreto. El resultado será hasta 1000 árboles en formato json con su información básica: creador, latitud, longitud y especie (si la tiene) y un enlace a la siguiente página (si hay más de 1000 árboles encontrados).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree?creator=cmayo`
B-T005	Ver detalles de un árbol	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri del árbol que se quiere consultar.	El usuario tiene que poder ver todos los datos de un árbol. El resultado será un json con toda la información acerca del mismo.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree/20200604-977df`
B-T006	Crear árbol	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.Es necesario estar registrado en el sistema.	El usuario registrado tiene que poder dar de alta árboles nuevos en el sistema. Para ello se tiene que autenticar mediante autenticación básica http e incluir, como mínimo, un json con la posición (latitud y longitud del árbol). Opcionalmente puede incluir en el json la especie y/o una imagen anotada (opcionales: título, descripción y parte que se visualiza). El resultado es un árbol y una o más anotaciones creadas en el sistema con identificadores únicos que contienen la fecha en formato YYYYMMDD.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-008:** / Ejemplo curl: `curl -k -H "Content-Type: application/json" -user cmayo:CristinaMayo1 -d '{"creator":"http://timber.gsic.uva.es/sta/data/user/cmayo", "lat": 50, "long': 1, "species":"http://timber.gsic.uva.es/sta/data/species/Species20", "depicts":"http://timber.gsic.uva.es/sta/ontology/Flower","title": "Árbol del jardín", "description": "Tomada en Valladolid", "image": '"$imagen_base64"' }' https://timber.gsic.uva.es/sta/data/tree/`
B-S001	Ver especies	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder consultar las especies disponibles en el sistema. El resultado será la taxonomía de especies del "Explotador Forestal" en formato json.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-009:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/species`
B-A001	Ver anotaciones creadas por un usuario	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el nombre de usuario del creador para el que se quiere consultar	El usuario tiene que poder consultar las anotaciones creadas por un usuario concreto. El resultado será las anotaciones de un creador en formato json con toda su información.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-006:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/annotation?creator=cmayo`

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado/Comentarios
B-A002	Ver detalles de una anotación	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri de la anotación que se quiere consultar.	El usuario tiene que poder ver todos los datos de una anotación. El resultado será un json con toda la información de esta.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-006:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/annotation/i-20200617-4dd87_c5a48`
B-A003	Crear anotación de tipo especie	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Para ello se tiene que autenticar mediante autenticación básica http e incluir, un json con el creador (debe coincidir con el de la autenticación), el árbol, el tipo (especie) y la especie del árbol (uri). El resultado es la creación de una anotación de tipo especie en el sistema y el establecimiento de esta especie como la primaria ("mejor"), en este momento, para el árbol.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Ejemplo curl: `curl -k -H "Content-Type: application/json" -user cmayo:CristinaMayo1 -d '{"creator":"http://timber.gsic.uva.es/sta/data/user/cmayo", "id": "http://timber.gsic.uva.es/sta/data/tree/20200601-e9058", "type":"species", "species":"http://timber.gsic.uva.es/sta/data/species/Species20"}' https://timber.gsic.uva.es/sta/data/annotation/`
B-A004	Crear anotación de tipo imagen	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Para ello se tiene que autenticar mediante autenticación básica http e incluir, un json con el creador (debe coincidir con el de la autenticación), el árbol, el tipo (imagen) y la imagen codificada en base64. El resultado es la creación de una anotación de tipo imagen en el sistema, el almacenamiento de la imagen en formato jpg y la relación de la anotación con el árbol.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Ejemplo curl: `curl -k -H "Content-Type: application/json" -user cmayo:CristinaMayo1 -d '{"creator":"http://timber.gsic.uva.es/sta/data/user/cmayo", "id": "http://timber.gsic.uva.es/sta/data/tree/20200601-e9058", "type":"image", "image": '"$imagen_base64"'}' https://timber.gsic.uva.es/sta/data/annotation/`
B-A005	Crear anotación de tipo posición	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Para ello se tiene que autenticar mediante autenticación básica http e incluir, un json con el creador (debe coincidir con el de la autenticación), el árbol, el tipo (posición) y la latitud y longitud del árbol. El resultado es la creación de una anotación de tipo posición en el sistema y el establecimiento de esta posición como la primaria ("mejor"), en este momento, para el árbol.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Ejemplo curl: `curl -k -H "Content-Type: application/json" -user cmayo:CristinaMayo1 -d '{"creator":"http://timber.gsic.uva.es/sta/data/user/cmayo", "id": "http://timber.gsic.uva.es/sta/data/tree/20200601-e9058", "type":"position", "lat": 42, "long": -3.8 }' https://timber.gsic.uva.es/sta/data/annotation/`
B-I001	Ver detalles de una imagen	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri de la imagen que se quiere consultar.	El usuario tiene que poder ver todos los datos de una imagen. El resultado será un json con toda la información acerca de la imagen.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-010:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/image/20200617-4dd87-24f44`
B-I002	Ver imagen	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri de la imagen que se quiere consultar.	El usuario tiene que poder ver cualquier imagen del sistema. El resultado será una imagen jpg.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-010:** / Hay dos formas de poder hacerlo: utilizando la cabecera `Accept=image/jpeg` sobre una url de tipo: `https://timber.gsic.uva.es/sta/data/image/20200617-4dd87-24f44` o consultando directamente el fichero: `https://timber.gsic.uva.es/data/images/20200617-4dd87-24f44.jpg`
B-I003	Ver todas las imágenes del sistema	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder ver una lista con todas las imágenes en el sistema, su fecha de creación y tamaño en bytes. Además puede seleccionar la que desee para visualizarla.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-010:** / A través de: `https://timber.gsic.uva.es/data/images/`
B-U001	Ver usuarios del sistema	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder ver una lista con todos los usuarios registrados en el sistema. El resultado será todos los usuarios almacenados y su información en formato json y un enlace a la siguiente página (si hay más de 1000 usuarios encontrados).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-003:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/user/`

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado/Comentarios
B-U002	Ver detalles de un usuario	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el identificador del usuario que se quiere consultar.	El usuario tiene que poder ver (públicos) de un usuario. El resultado será un json con toda la información acerca del mismo.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-004:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/user/cmayo`
B-U003	Crear usuario	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	Un usuario tiene que ser capaz de poder registrarse en el sistema. Para ello tiene que introducir el nombre completo, email y contraseña en un json y elegir un identificador o nombre de usuario no utilizado. El resultado será un nuevo usuario creado en el sistema.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-001:** / Ejemplo curl: `curl -k -X PUT -H "Content-Type: application/json" -i https://timber.gsic.uva.es/sta/data/user/tel_uva_forestal20 -d '{"nombre":"Cristina Mayo Sarmiento", "email": "teluva_forestal@tel.uva.es", "password":"Pass1234" }'`
B-TP001	Ver posibles partes de un árbol en una foto	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder ver todos las posibles partes definidas para etiquetar en una imagen. El usuario podrá elegir cualquiera de estas opciones a la hora de determinar qué se muestra en una imagen de un árbol. El resultado será un json con todos los elementos definidos en la ontología creada para la aplicación (hoja, fruto, tronco, rama, copa, flor, vista general, otra parte).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	Ejemplo: `https://timber.gsic.uva.es/sta/data/treePart`

Tabla 6.1: Pruebas funcionales: casos de prueba *software*. Flujos básicos.

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado / Comentarios
B-T001-1	Ver árboles del sistema (error de parámetro)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder consultar los árboles del sistema. Si se incluye un parámetro de consulta no existente, el sistema lo obviará el resultado será hasta 1000 árboles en formato json con su información básica: creador, latitud, longitud y especie (si la tiene) y un enlace a la siguiente página (si hay más de 1000 árboles encontrados).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree?tipo=flor`
B-T001-2	Ver árboles del sistema (no hay datos)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder consultar los árboles del sistema. Si no hay ningún árbol el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree?tipo=flor`
B-T002-1	Ver árboles de una zona (error de parámetro)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer las coordenadas del área que se quiere consultar.	El usuario tiene que poder consultar los árboles en una zona delimitada por dos latitudes y dos longitudes cualesquiera. Si el parámetro de consulta de una de las cuatro longitudes es incorrecto o inexistente, el resultado será un mensaje de error (código 400) indicando que la petición es errónea.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:** / Ejemplo: `https://timber.gsic.uva.es/sta/data/tree?lat0=41.66&long0=-4.72&long1=-4.7239`
B-T003-1	Ver árboles de una especie (no hay datos)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer la especie que se quiere consultar.	El usuario tiene que poder consultar los árboles de una especie concreta. Si no hay ningún árbol de esa especie el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:**
B-T004-1	Ver árboles creados por un usuario (no hay datos)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el nombre de usuario del creador para el que se quiere consultar.	El usuario tiene que poder consultar los árboles de un creador concreto. Si no hay ningún árbol creado por un usuario concreto el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:**
B-T004-2	Ver árboles creados por un usuario (usuario no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el nombre de usuario del creador para el que se quiere consultar.	El usuario tiene que poder consultar los árboles de un creador concreto. Si el creador no existe como usuario en el sistema el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:**
B-T005-1	Ver detalles de un árbol (no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri del árbol que se quiere consultar.	El usuario tiene que poder ver todos los datos de un árbol. Si el árbol no existe el resultado será un mensaje de error (código 404) indicando que el árbol no existe.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-007:**
B-A001-1	Ver anotaciones creadas por un usuario (no hay datos)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el nombre de usuario del creador para el que se quiere consultar	El usuario tiene que poder consultar las anotaciones creadas por un usuario concreto. Si no hay anotaciones para ese usuario el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-006:**
B-A001-2	Ver anotaciones creadas por un usuario (usuario no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el nombre de usuario del creador para el que se quiere consultar	El usuario tiene que poder consultar las anotaciones creadas por un usuario concreto. Si el creador no existe como usuario en el sistema el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-006:**

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado / Comentarios
B-A002-1	Ver detalles de una anotación (no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri de la anotación que se quiere consultar.	El usuario tiene que poder ver todos los datos de una anotación. Si la anotación no existe el resultado será un mensaje de error (código 404) indicando que la anotación no existe.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-006:**
B-I001-1	Ver detalles de una imagen (no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri de la imagen que se quiere consultar.	El usuario tiene que poder ver todos los datos de una imagen. Si la imagen no existe el resultado será un mensaje de error (código 404) indicando que la imagen no existe.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-010:**
B-I002-1	Ver imagen (no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el uri de la imagen que se quiere consultar.	El usuario tiene que poder ver cualquier imagen del sistema. Si la imagen no existe el resultado será dun mensaje de eror (código 404) de recurso no encontrado.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-010:** / No es propiamente de la API REST, en realidad es contenido estático servido pro el Nginx.
B-U001-1	Ver usuarios del sistema (no hay datos)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.	El usuario tiene que poder ver una lista con todos los usuarios registrados en el sistema. Si no hay ningún usuario el resultado será un mensaje vacío (código 204).	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-003:**
B-U002-1	Ver detalles de un usuario (no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario conocer el identificador del usuario que se quiere consultar.	El usuario tiene que poder ver todos los datos de un usuario. Si el usuario no existe el resultado será un mensaje de error (código 404) indicando que el usuario no existe.	Cristina Mayo Sarmiento	Firefox Mozilla 77.0.1 (64-bit) en Windows 10/curl 7.58.0	01/07/2020	✓	**CU-004:**
B-T006-1	Crear árbol (error de autenticación: usuario incorrecto)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.Es necesario estar registrado en el sistema.	El usuario registrado tiene que poder dar de alta árboles nuevos en el sistema. Si el nombre de usuario es incorrecto el resultado será un mensaje de error (código 401) indicando *login* incorrecto.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-008:**
B-T006-2	Crear árbol (error de autenticación: contraseña incorrecta)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.Es necesario estar registrado en el sistema.	El usuario registrado tiene que poder dar de alta árboles nuevos en el sistema. Si la contraseña de usuario es incorrecta el resultado será un mensaje de error (código 401) indicando contraseña incorrecta.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-008:**
B-T006-3	Crear árbol (error de autenticación)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio.Es necesario estar registrado en el sistema.	El usuario registrado tiene que poder dar de alta árboles nuevos en el sistema. Si el usuario no se identifica en el sistema el resultado será un mensaje de error (código 401) indicando que falta la cabecera de autenticación.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-008:**
B-A003-1	Crear anotación de tipo especie (error de autenticación: usuario incorrecto)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Si el nombre de usuario es incorrecto el resultado será un mensaje de error (código 401) indicando *login* incorrecto.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Análogo para B-A004 y B-A005
B-A003-2	Crear anotación de tipo especie (error de autenticación: contraseña incorrecta)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Si la contraseña de usuario es incorrecta el resultado será un mensaje de error (código 401) indicando contraseña incorrecta.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Análogo para B-A004 y B-A005

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado / Comentarios
B-A003-3	Crear anotación de tipo especie (error de autenticación)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Si el usuario no se identifica en el sistema el resultado será un mensaje de error (código 401) indicando que falta la cabecera de autenticación.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Análogo para B-A004 y B-A005

Id	Caso de prueba	Prerrequisitos	Descripción	Usuario	Navegador /curl	Fecha	Éxito/ Fallo	CU Asociado / Comentarios
B-A003-4	Crear anotación de tipo especie (árbol no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Si el árbol al que se asocia la anotación no existe en el sistema el resultado será un mensaje de error (código 400) indicando que el árbol no existe.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Análogo para B-A004 y B-A005
B-A003-5	Crear anotación de tipo especie (creador no existe)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Si el creador no existe o no coincide con la autenticación el resultado será un mensaje de error (código 401) indicando que el *login* es incorrecto.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✓	**CU-005:** / Análogo para B-A004 y B-A005
B-A003-6	Crear anotación de tipo especie (faltan campos obligatorios)	El servicio tiene que estar ejecutándose y es necesario conectarse mediante *proxy* o desde la red del laboratorio. Es necesario estar registrado en el sistema y seleccionar un árbol que exista previamente.	El usuario registrado tiene que poder dar de alta anotaciones en el sistema. Si falta algún campo obligatorio (latitud, longitud o tipo) el resultado será un mensaje de error (código 400) indicando que faltan campos obligatorios.	Cristina Mayo Sarmiento	curl 7.58.0	01/07/2020	✗	**CU-005:** / Análogo para B-A004 y B-A005 con sus respectivos campos obligatorios. Falta validación.

Tabla 6.2: Pruebas funcionales: casos de prueba *software*. Flujos alternativos.

6.3 Pruebas de carga y comparativa de rendimiento

El rendimiento y las métricas de un sistema *software* y, concretamente, de una API REST se pueden obtener mediante multitud de herramientas. Para este trabajo se ha decidido usar el módulo JavaScript `loadtest` [69] por su compatibilidad y facilidad de integración con Node.js y Express. Básicamente permite ejecutar pruebas de manera automática sobre la API REST –una vez se han configurado los parámetros deseados– simulando un número de usuarios concurrentes en el sistema y un tiempo máximo de ejecución de las pruebas. La herramienta ofrece varios resultados como los tiempos en milisegundos que tarda en resolverse cada petición, el número de peticiones ejecutadas (finalizadas con éxito y fallidas) y un resumen con datos como el valor medio de resolución de las peticiones, entre otros.

Previamente a la ejecución de las pruebas se ha poblado el sistema con datos suficientes para que sea lo más realista posible, desde el punto de vista de escenarios con cierto nivel de exigencia. El número de triplas en el Virtuoso antes de la ejecución de las pruebas es 437.836, existen 18.539 árboles, 91 usuarios registrados, 69.197 anotaciones y 10.005 imágenes cuyos ficheros JPG ocupan 1,15 GB en el sistema de ficheros del servidor. Para las pruebas las imágenes que se añaden al crear un árbol o una anotación de tipo imagen tienen un tamaño de 147KB.

Se empiezan las pruebas con el servicio recién reiniciado para que no haya datos cacheados inicialmente y se realizan pruebas de rendimiento de las operaciones que requieren más carga o son más complejas, obviando aquellas como ver los detalles de un árbol o anotación concretos, que si bien serán algo más lentas cuando la caché esté vacía (para una prueba manual, a través de Firefox, con un usuario se resuelve en 115ms al iniciarse el servicio y el almacén de triplas), posteriormente se resolverán rápidamente en tiempos inferiores a 30ms (para un usuario). Tampoco se incluyen en las pruebas de carga las creaciones de usuarios porque la librería no permite parametrizar distintos identificadores para generar usuarios distintos (método PUT) para usuarios concurrentes. De todas formas, estas creaciones no son críticas ya que que incluyen JSON reducidos y las consultas de inserción de triplas son mucho más sencillas que en el caso de crear un árbol o anotación. Típicamente, de media, una petición de creación de usuario se resuelve en 53 ms (dato obtenido mediante pruebas con un usuario).

Bajo estas consideraciones, se realizan test de rendimiento del *back-end*, lanzando peticiones a los distintos métodos de la API REST de forma incesante durante 30 segundos para cada una de las pruebas[1] y con distinto nivel de concurrencia (es decir cierto número de clientes simultáneos que se ejecutan en paralelo) para las operaciones más exigentes. Estos valores se han seleccionado de este modo como un compromiso entre el tiempo que tardan en ejecutarse las pruebas y el valor que ofrecen los resultados para su estudio y su posible extrapolación a otras situaciones y analogías con escenarios de uso real (con usuarios).

Los resultados de los tiempos medios de ejecución para cada prueba y las pruebas concretas se muestran en la Tabla 6.3, con un total de 42.983 peticiones ejecutadas en un tiempo total de 13 minutos y medio (lo que puede considerarse un uso intensivo del sistema). Cabe destacar que al ser pruebas con cargas sintéticas el comportamiento del sistema no será idéntico a un caso con carga real. Por ejemplo, en un escenario real es difícil que exactamente en el mismo instante múltiples usuarios estén creando datos simultáneamente sobre el mismo árbol o consultando exactamente la misma información. Debido a esto,

[1] Cada prueba se denomina a todo lo que engloba a las peticiones que realizan sobre un método de la API REST.

	Concurrencia=1	Concurrencia=5	Concurrencia=10
Petición	**Tiempo medio por petición (ms)**	**Tiempo medio por petición (ms)**	**Tiempo medio por petición (ms)**
Crear árbol completo	170	450	900
Crear anotación de especie para árbol existente	140	380	750
Crear anotación de posición para árbol existente	140	380	750
Crear anotación de imagen para árbol existente	160	400	810
Ver árboles del sistema (1000 resultados)	20	50	100
Ver árboles en un área (100 resultados)	40	100	190
Ver árboles en de una especie (100 resultados)	20	50	90
Ver árboles de un usuario (100 resultados)	10	20	60
Ver anotaciones de un usuario (350 resultados)	20	30	100
Total de peticiones ejecutadas	7.519	17.576	17.888

Tabla 6.3: Pruebas de carga ejecutadas con `loadtest` para 30 segundos de experimento.

la concurrencia elegida es relativamente pequeña (hasta 10 usuarios), aunque se espera que pueda haber al mismo tiempo más usuarios y no supondría ningún problema para el servicio, puesto que, la exigencia será menor porque la concurrencia y simultaneidad no serán tales. Otra diferencia entre la situación asociada a la carga sintética y un escenario con carga real es que las peticiones que recibe el servidor en cada momento son variadas, pero la herramienta solo permite realizar las pruebas de forma secuencial. Se puede señalar que la creación de anotaciones de posición y especie es más lenta de lo que se podría esperar – se esperaban tiempos con diferencias significativamente menores que los asociados creación de árboles, sin embargo son comparabales. La justificación se haya en que al crear una nueva anotación, en la implementación actual del prototipo, se actualizan las anotaciones primarias de especie y posición del árbol y, además, se están creando continuamente anotaciones sobre el mismo árbol.

Los resultados de los tiempos necesarios para resolver cada petición en milisegundos bajo las condiciones especificadas anteriormente y un nivel de concurrencia de 5 usuarios se pueden representar en diagramas de caja o *boxplot* con el objetivo de obtener una visión estadística de estos. Este valor de concurrencia, aunque puede parecer pequeño, puede considerarse un valor medio frecuente en situaciones con usuarios reales, ya que es muy difícil que exactamente en el mismo instante haya un número más elevado de usuarios simultáneos realizando peticiones al sistema. En la Figura 6.1 se muestran los *boxplots* para los métodos `POST` de la API REST, asociados a la creación de árboles y anotaciones. Lógicamente, los tiempos más grandes se obtienen al crear un árbol completo y al crear anotaciones de imagen, ya que son las peticiones con mayor carga de datos y que más operaciones requieren. Aunque hay algunos *outliers*, no son demasiados, ya que los valores no tienden a ser atípicos. En las Figuras 6.2 y 6.3, se muestran sendos diagramas para los métodos `GET`. Como se puede observar el rendimiento del sistema es mucho menor al arranque de este (Figura 6.2), ya que las cachés aún no están pobladas y es necesario consultar los datos del Virtuoso. Disminuyen notablemente los tiempos necesarios para resolver las peticiones (estas peticiones ya han sido realizadas anteriormente, por lo que están cachéadas) tras un tiempo de experimento del sistema en funcionamiento (Figura 6.3), al utilizarse la caché implementada en la API REST como un objeto JavaScript y la asociada al módulo `memory-cache` (recordar que esta última cachea peticiones por su URL y es especialmente eficiente cuando las peticiones se repiten en un intervalo de tiempo reducido). Los *boxplots* asociados a las peticiones GET presentan múltiples *outliers* debido, generalmente, a la influencia de las cachés y al encolado de peticiones (son asíncronas, lo que quiere decir que el sistema no se bloquea al recibir una petición). Cuando no se tienen los datos o las peticiones no están cacheadas los tiempos de respuesta son mayores (y atípicos, ya que se dan en caso de peticiones nuevas o no recientes),

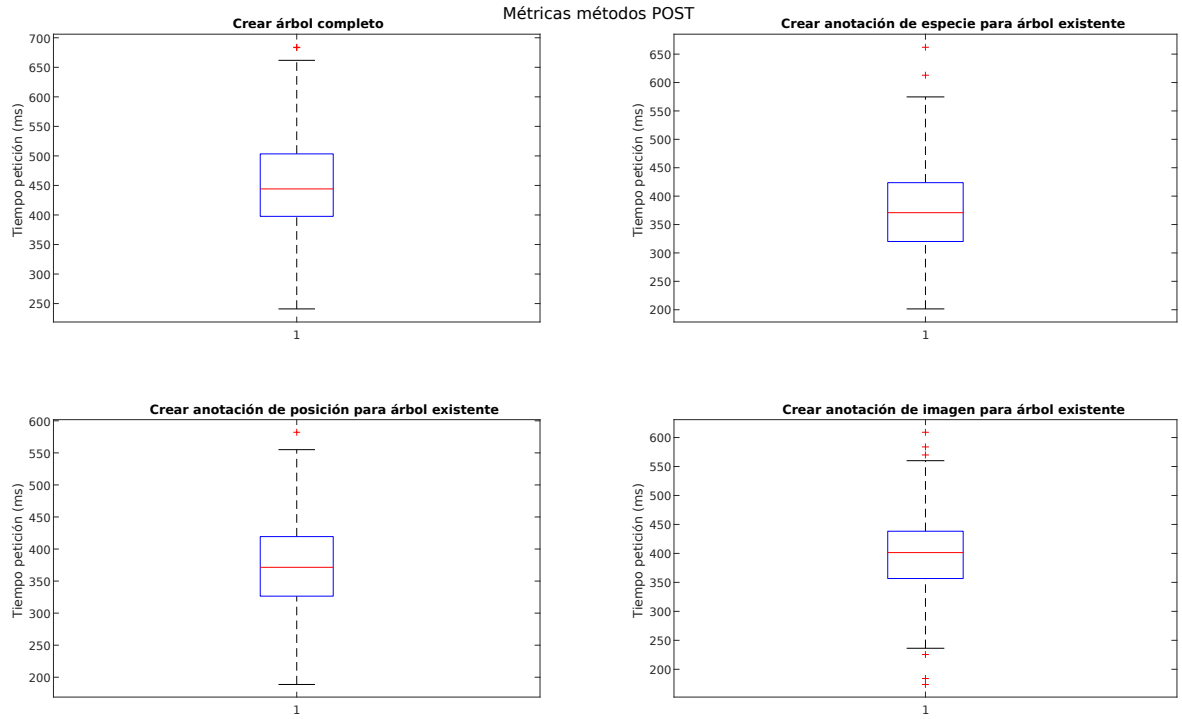


Figura 6.1: *Boxplots* de métodos `POST` con `loadtest` para 30 segundos de experimento y 5 usuarios concurrentes.

mientras que en el caso contrario los tiempos son más pequeños. Además, en este tipo de pruebas en las que se "bombardea" al sistema con peticiones y este las resuelve de forma asíncrona, no es extraño que los tiempos de respuesta presenten múltiples valores atípicos. Proporcionan una visión del comportamiento del sistema en situaciones extremas como puede ser la saturación o infrautilización. Estas casuísticas se reflejan en el conjunto de datos como valores atípicos por encima o por debajo de los valores mínimo y máximo, respectivamente.

Con el objetivo de comprobar cómo escala el sistema según crece el número de usuarios y obtener un nivel de referencia –el sistema está siendo utilizado por un único usuario– se han realizado múltiples pruebas de carga como las presentadas anteriormente, pero con un número de usuarios concurrentes distinto a 5, siendo 1 y 10 los valores elegidos. Se han representado los tiempos de respuesta para cada uno de los experimentos realizados en gráficas en una misma figura, a modo de comparativa. Se incluyen los *boxplots* para 1, 5 y 10 usuarios concurrentes en cada una de ellas. Para las peticiones de consulta estos experimentos se duplican, ya que se realizan para consultas sin cachear (arriba) y cacheadas (abajo). Las gráficas para cada experimento se ilustran en el Apéndice F. Se puede destacar que la mediana de los tiempos de respuesta crece con el número de usuarios concurrentes, así como los *outliers*. El número de usuarios concurrentes afecta más en los tiempos de respuesta de los métodos POST, relacionados con las creaciones, que en los métodos GET, de consultas. En el caso de peticiones repetidas y, por tanto, cacheadas, el número de *outliers* se reduce ya que recuperar los datos de memoria es mucho más rápido

y los tiempos necesarios serán más uniformes.

A la vista de los resultados se puede concluir que se satisfacen los requisitos no funcionales (ver apartado 3.3.2) de resolver las peticiones en tiempos inferiores a 1 segundo para las lecturas y de 5 segundos para las creaciones, por lo que se puede dar servicio a un número de usuarios –incluso concurrentes– superior a 10. La operación más exigente es la creación de un árbol anotándolo completamente (posición, especie e imagen), no obstante las operaciones se esperan más frecuentes en un entorno de producción son las de tipo consulta, ya que no se requiere estar registrado en el sistema como usuario. Se estima que las creaciones supondrán en torno a un 1-10 % del total de las peticiones, aproximadamente.

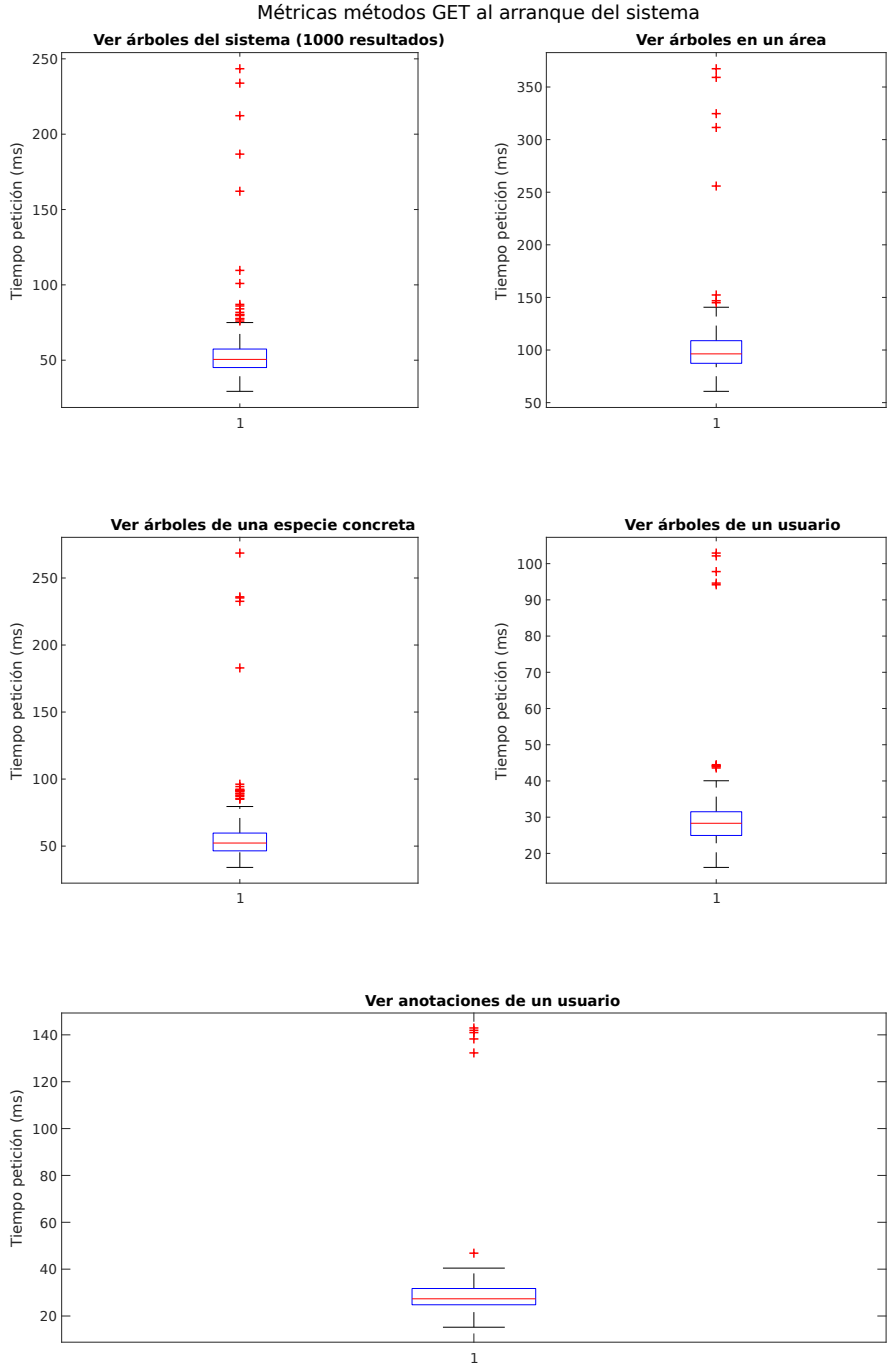


Figura 6.2: *Boxplots* de métodos `GET` al inicio del sistema con `loadtest` para 30 segundos de experimento y 5 usuarios concurrentes.

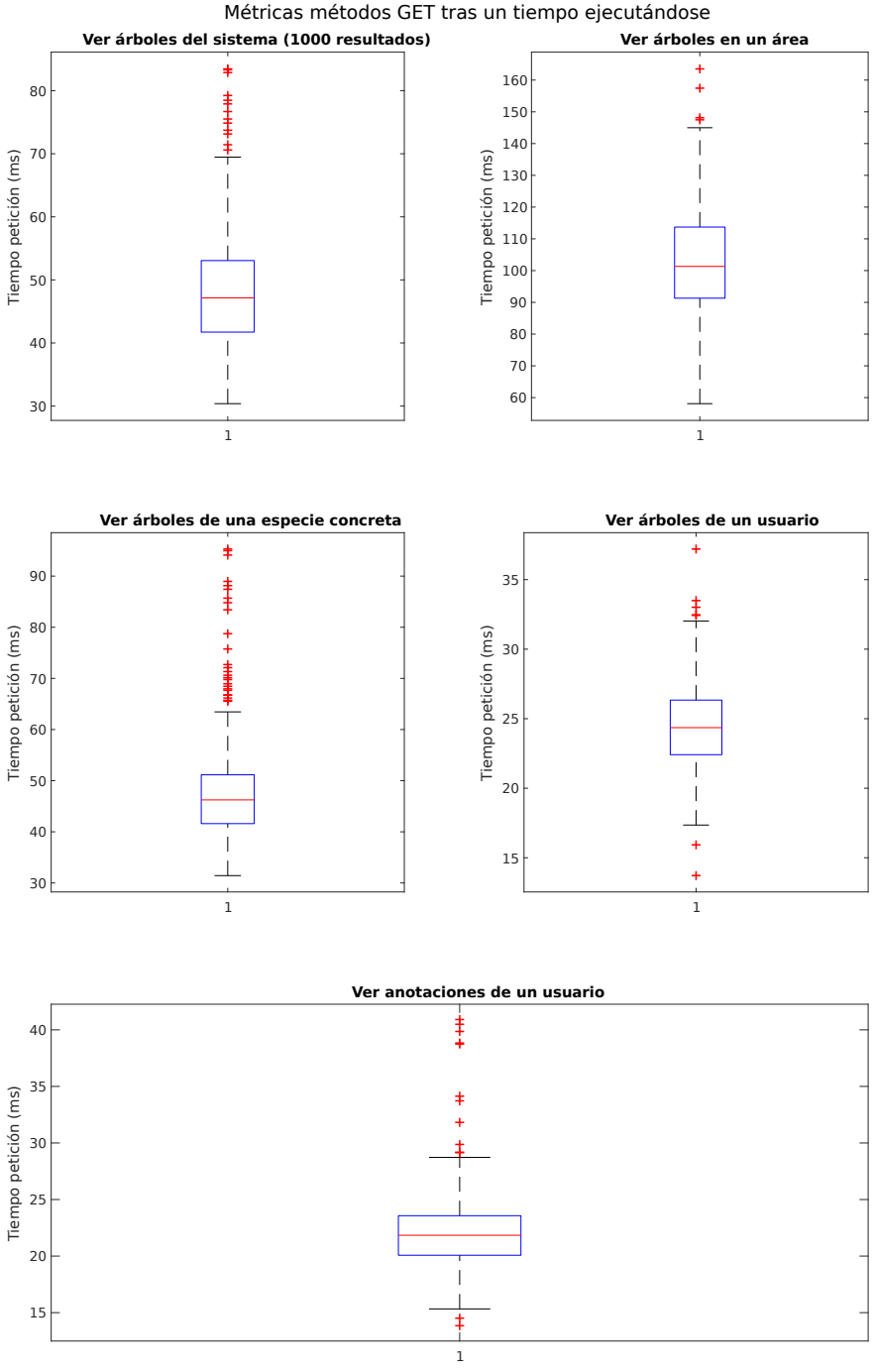


Figura 6.3: *Boxplots* de métodos `GET` tras un tiempo de ejecución (peticiones cacheadas) con `loadtest` para 30 segundos de experimento y 5 usuarios concurrentes.

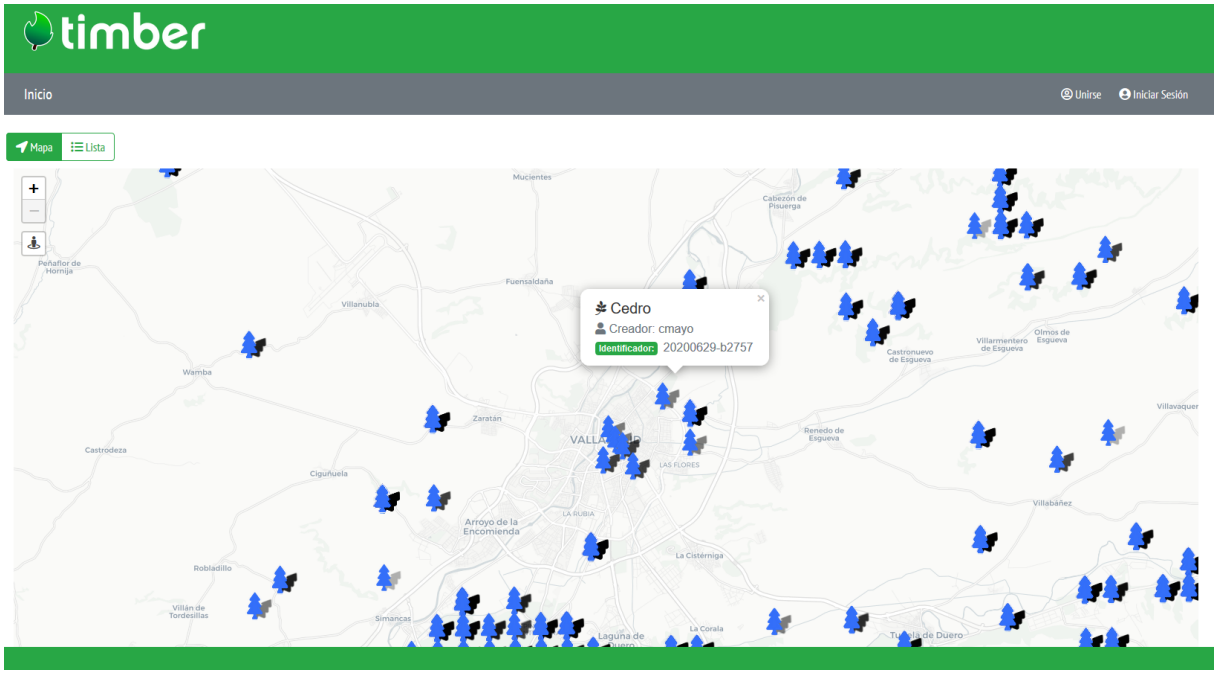
6.4 Integración con un front-end

Una de las ventajas de desarrollar el sistema con una arquitectura como la que se ha planteado, con el *back-end* separado del *front-end*, es la versatilidad que ofrece, dado que una API REST se puede integrar con multitud de aplicaciones cliente: aplicaciones móviles, aplicaciones web, interfaces de comandos, etc. La puesta en marcha combinada de los dos sistemas (*back-end* y *front-end*) como uno solo resulta sencilla, ya que el *back-end* es completamente independiente del *front-end*. A su vez, para cualquier *front-end* que se quiera integrar, es suficiente con conocer los recursos expuestos, los métodos existentes para cada recurso, los datos que se reciben o envían y el formato de los mismos. Toda está información (incluyendo parámetros, tipo de datos, formatos...) se encuentra recogida en la correspondiente documentación de la API (ver ApéndiceB), por lo que no tendría que ser problemático para ningún desarrollador.

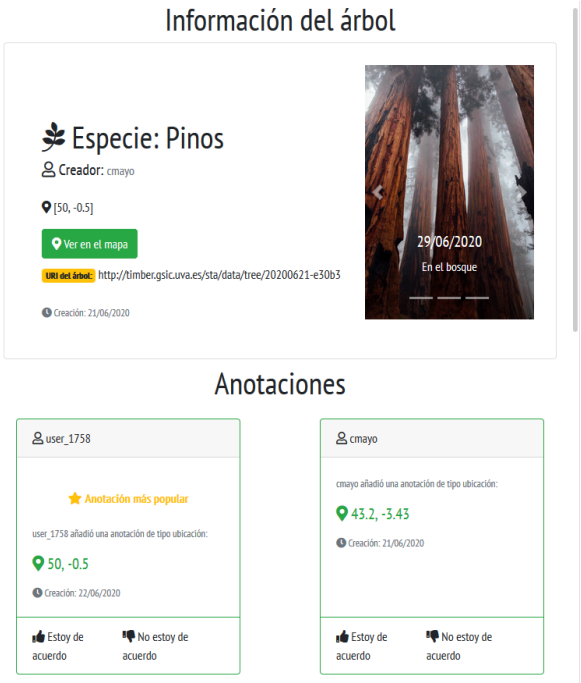
A continuación, se muestra un ejemplo concreto de una aplicación web con Angular desarrollada por [36] que se integra con el servidor implementado en este Trabajo Fin de Máster y consume la API REST ofrecida. En la Figura 6.4 se muestran capturas de algunas de las pantallas de esta aplicación, con el objetivo de dar una visión de lo que podría considerarse la aplicación completa de anotación de árboles. Esta aplicación es adecuada para usuarios finales sin conocimientos técnicos y su desarrolladora [36] ha verificado las afirmaciones anteriores respecto a la facilidad de integración con el *back-end* son reales.

6.5 Conclusiones

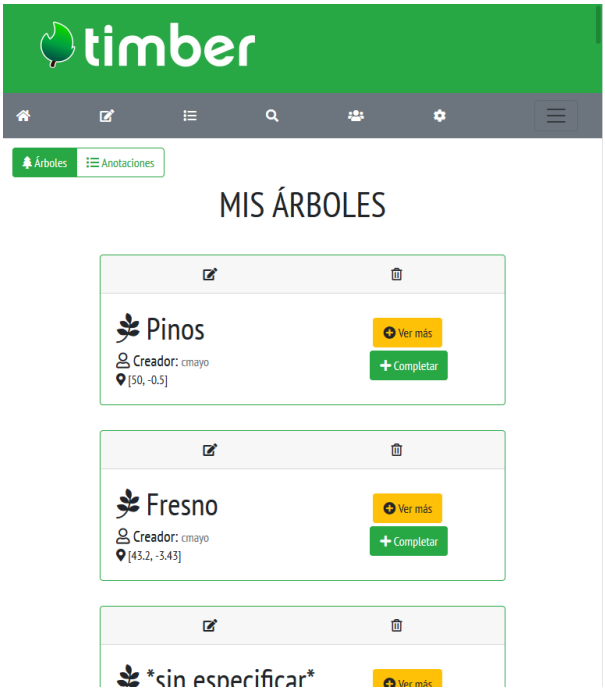
El presente capítulo ha recogido las actividades relacionadas con la validación de los procesos de desarrollo de *software*. La realización de las distintas pruebas ha permitido evaluar el sistema a nivel funcional, es decir, demostrar que satisface los requisitos funcionales implementados propuestos en el capítulo 3, así como los casos de uso. Para ello se han diseñado múltiples casos de prueba que han permitido sistematizar las pruebas a realizar. A nivel de rendimiento, mediante pruebas automáticas y cargas sintéticas, se consigue cuantificar el funcionamiento del sistema, en relación al tiempo de respuesta de los distintos tipos de peticiones con distinto número de usuarios y las influencias de las cachés. Como era de esperar, las peticiones de creación son más lentas que las consultas, aunque se estima que estas sean muy inferiores (en torno al 10 % del total de peticiones que reciba el sistema). Las consultas cacheadas son muy eficientes al evitar comunicaciones con el Virtuoso y son especialmente rápidas las peticiones repetidas sobre la misma URL en una ventana de tiempo pequeña (por ejemplo, aplica cuando hay varios usuarios realizando las mismas consultas de forma concurrente). Se puede concluir que el sistema presenta un rendimiento adecuado con distinto número de usuarios concurrentes y peticiones, ya que los resultados obtenidos satisfacen los requisitos no funcionales. A nivel de integración con otras aplicaciones de tipo *front-end*, las pruebas realizadas han sido con una aplicación web Angular [36] y han resultado satisfactorias, proporcionando una visión global de la posible la aplicación resultante.



(a) Pantalla principal.



(b) Vista de los detalles de un árbol.



(c) Pantalla de los árboles de un usuario registrado.

Figura 6.4: Ejemplos de pantallas de una aplicación web Angular [36] integrada con el *back-end* desarrollado.

Capítulo **7**

Conclusiones y líneas futuras

El capítulo final del documento tiene como objetivo resaltar las principales conclusiones extraídas de la realización de este Trabajo Fin de Máster, así como indicar las limitaciones presentes en el *back-end* desarrollado y las posibles líneas de trabajo futuras.

7.1 Conclusiones

La Web es uno de los principales medios de distribución de contenidos, usado de forma masiva por millones de personas a través de distintos dispositivos. Las principales razones son el volumen de datos almacenados y la facilidad de acceso. No obstante, la publicación de datos en la Web presenta una serie de desafíos y dificultades relacionados con la heterogeneidad de los datos, el uso de distintos formatos, los modelos de datos dispares o incompatibles que imposibilitan la integración, las inconsistencias o la mala documentación. Además convierte, muy frecuentemente, a datos útiles en información difícilmente reutilizable y favorece la fragmentación de estos y la creación de silos aislados en la Web. Por tanto, se requieren nuevas formas de exponer, compartir y conectar datos en la Web como LOD.

El origen de los datos existentes en la Web es muy variado: datos abiertos gubernamentales, científicos o procedentes de medios sociales. Se pone de manifiesto el cambio de rol de los usuarios de consumidores pasivos de contenidos a creadores debido a la popularidad de los sitios web sociales. Esto ha provocado una gran cantidad de datos sociales existentes en la Web en la actualidad, generados por usuarios. Asimismo, ha habido un incremento de datos abiertos publicados en la Web por parte de gobiernos. Esta cantidad de información puede utilizarse con múltiples propósitos.

En el área forestal, la cantidad de datos que se manejan es muy elevada y en este sector cada vez se usan más datos para gestión y planificación forestal, entre otras aplicaciones. A su vez, existen numerosas iniciativas de datos abiertos relacionadas. Esta información, recogida por ingenieros forestales y mantenida por organismos públicos se almacena en inventarios forestales de acceso público. Sin embargo, a pesar de que los datos suelen ser muy precisos y están disponibles para cualquiera que desee consultarlos, tienden a ser datos aislados, en formatos desconocidos y cuya consulta no es trivial para otros interesados como senderistas, estudiantes o aficionados. En el caso del España, se utiliza Microsoft Access para las bases de datos del IFN, una decisión tecnológica cuestionable en tanto que no es adecuada para grandes volúmenes de datos ni es multiplataforma. El formato del MFE es shapefile, el cual requiere 3 o más ficheros para almacenar los datos con distintas extensiones, es muy limitado en cuanto a almacenamiento de datos, formatos de los datos, campos y atributos y es anticuado. Al mismo tiempo, los datos están

restringidos a zonas de interés debido al coste que supone la toma de datos y la elaboración de inventarios a las administraciones públicas. Se actualizan periódicamente (cada década en el caso del IFN), por lo que no siempre son actuales.

La aplicación de tecnologías de la Web Semántica en el ámbito forestal permite la publicación de los datos de forma abierta y accesibles al público general, vincularlos con ontologías relacionadas con territorios geográficos y políticos o de descripción de especies. El resultado de este tipo de publicación beneficia a organismos gubernamentales y no gubernamentales, empresas y particulares. Tiene como ventajas la facilidad de integración de datos y la flexibilidad, ya que permite datos heterogéneos o variables, la reutilización (permite descargar los datos en formatos estándares) y la colaboración. Cabe destacar que el principal beneficio de la Web Semántica es la integración de datos, por ejemplo, integración de datos de distintos tipos y fuentes, inventarios de diferentes ediciones, la conexión con datos de cobertura arbórea, el etiquetado de especies... Del mismo modo, el uso de estas tecnologías permite realizar consultas más complejas sobre los datos de forma sencilla. Junto con el uso de ontologías comunes se simplifican las consultas al poder utilizar términos sin ambigüedad.

En este Trabajo Fin de Máster se ha planteado el diseño e implementación del servidor de una aplicación social de anotación de árboles que utilice las tecnologías de la Web Semántica con el fin de intentar aprovechar la "inteligencia colectiva" y generar nuevos datos forestales de manera gratuita y útil para la comunidad. Mediante esta aplicación se pueden crear nuevos árboles no existentes en los inventarios oficiales y proponer sugerencias de actualización de los datos oficiales del IFN. Esta información se construye a partir de anotaciones que hacen uso de una ontología de árboles diseñada con los términos necesarios para la misma y con otras ontologías bien conocidas para facilitar la reutilización de los datos. La información generada por la aplicación está publicada como LOD en la Web de Datos de forma estructurada, mediante identificadores únicos para facilitar y permitir consultas y reutilización sin restricciones. Estos datos son heterogéneos ya que, además de información textual relativa a posiciones y especies de los árboles, se añaden imágenes anotadas y etiquetadas.

Esta iniciativa social de ciencia ciudadana es una propuesta novedosa dado que no se han encontrado referencias de otras alternativas con estas funcionalidades. Permite aprovechar el conocimiento colectivo para una tarea no trivial como es el etiquetado de especies y, posteriormente, usar esta información en ámbitos educativos para fomentar el aprendizaje o simplemente como contribución *crowdsourcing* para anotar más especies de árboles. Por otro lado, permite la consulta de datos oficiales mediante la integración de datos en RDF del IFN convertidos por el proyecto europeo Cross-Forest con la aplicación. Se explotan, por tanto, las tecnologías de la Web Semántica, RDF, RDFS, OWL y SPARQL, para poder gestionar adecuadamente gran variedad de datos, integraciones y heterogeneidad (datos institucionales mezclados con datos de usuario, posiciones, imágenes...) de forma sencilla.

Existen tres partes principales que se pueden destacar en este trabajo: la ontología creada, los datos generados y expuestos y la API REST implementada. Respecto al modelo de datos, además de reutilizar ontologías existentes, se ha creado una ontología propia. Esta define una serie de términos que se utilizan para anotar los árboles evitando ambigüedades y permitiendo que los datos sean procesables computacionalmente. Esto ofrece múltiples ventajas relacionadas con la flexibilidad, extensibilidad e integración con otras fuentes de datos. Asimismo una ontología normalmente es más comprensible que un esquema de una base de datos relacional, ya estos últimos típicamente están diseñados para optimizar su rendimiento,

dejando a un lado la facilidad de interpretación.

Aparte del modelo de datos creado, los datos se ofrecen como LOD en la Web. Se integran datos institucionales del IFN con datos sociales de los usuarios, los cuales se pueden diferenciar gracias a los términos que incluye la ontología. De esta forma la aplicación, además de generadora de datos, sirve como un punto de consulta de acceso sencillo para cualquier usuario. Los usuarios puede evaluar la fiabilidad de los datos en función de cómo estén anotados en la ontología. Se construye un punto de confluencia de conocimiento entre datos oficiales y no oficiales, validados por expertos o con mejor puntuación social y se contribuye a la generación de contenido a la Web de Datos.

En el mismo orden de ideas, en este trabajo hay que destacar el diseño e implementación de una API REST que permite que los datos sean fácilmente utilizados por cualquiera. La principal función de la API es enmascarar la complejidad de las tecnologías semánticas subyacentes a los posibles *front-end* (aplicaciones web, móviles...) que se integren con el *back-end* semántico que conforma la aplicación. Es importante señalar que, a pesar de la potencia que tienen la Web Semántica y LOD, su comprensión y la utilización, así como el conocimiento de sus principios y fundamentos requiere una formación extensa y ardua. Por estos motivos el valor de la API debe recalcarse. Fundamentalmente expone métodos y recursos bien documentados y proporciona representaciones de los recursos en formatos estándar, actuando como *proxy* entre los recursos expuestos y el almacén de triplas. Traduce las peticiones HTTP en SPARQL, recupera los datos en forma de triplas y los formatea adecuadamente para los clientes.

Por último, considerando el servidor diseñado e implementado como un todo, y en estrecha relación con la arquitectura propuesta, las tecnologías elegidas y la implementación propiamente dicha, se ha comprobado la importancia del uso de cachés y funciones asíncronas para la API, cuyo comportamiento tiene un fuerte impacto en el funcionamiento general del sistema. Las cachés han permitido que el sistema resuelva un número muy superior de peticiones, ya que el tiempo de resolución de estas es mucho menor que en el caso de no disponer de ellas, y evita repetir consultas al almacén de triplas. Esto se ha podido cuantificar mediante experimentos sobre el sistema con cargas sintéticas que, además, han permitido evaluar la escalabilidad del sistema cuando aumenta el número de usuarios concurrentes. Por otro lado, el uso de Node.js y Express para la implementación de la API ha posibilitado una implementación más sencilla que con otras alternativas como Restlet, ofreciendo un buen rendimiento mediante el uso de funciones asíncronas y con baja latencia, de forma que es capaz de atender un número mayor de usuarios o peticiones y minimizar las esperas, ya que el sistema no se bloquea al recibir cada petición. Respecto a la validación de la funcionalidad del sistema, la sistematización de las pruebas mediante los casos de prueba ha facilitado la comprobación del comportamiento del servidor tanto en situaciones esperadas, siguiendo los flujos básicos, como en escenarios excepcionales a través de flujos alternativos, destacando el tratamiento de los errores por parte de la API en estos casos.

Por todas estas razones se concluye que el *back-end* presentado está disponible y es apto para su integración con cualquier *front-end* que haga uso de la API expuesta. Se ofrece un servicio fluido a los usuarios interesados en temática forestal, permitiendo anotaciones básicas de posiciones, especies e imágenes de árboles y etiquetando e identificando partes de las mismas de manera semántica. Por añadidura, se pueden consultar datos del IFN relativos a árboles y sus posiciones y especies de forma simple.

7.2 Limitaciones y líneas futuras

Pese a que el análisis del sistema establece requisitos y casos de uso relacionados con la parte más "social" de la aplicación como la existencia de varios roles de usuarios, la valoración de anotaciones de otros usuarios y la supervisión, en el análisis del modelo de datos y en el diseño e implementación del *back-end* no se han incluido estas funcionalidades. Por tanto, se podría ampliar la ontología para incorporar estos requisitos y extender la API y la lógica subyacente para soportarlo. Al explotar la parte social, el conocimiento colectivo obtenido sería mayor y podría contribuir al curado de los datos. Actualmente, la curación de los datos recae sobre el administrador y/o desarrollador del sistema, ya que no hay mecanismos para comprobar que los datos que introducen los usuarios son verídicos o fiables. Sin embargo, con el sistema de valoraciones y supervisión planteado las anotaciones primarias se corresponderían con los datos más aceptados por la comunidad y las anotaciones afirmadas en datos validados por expertos. En esta línea, también podría resultar interesante la integración de un servicio de reconocimiento de imágenes para eliminar aquellas inadecuadas. Incluso, se puede plantear la integración con alguna tecnología de *machine learning* para la identificación automática o semiautomática de especies de árboles a partir de imágenes. En relación a la identificación y anotado de partes de una imagen (por ejemplo, señalar una flor concreta de un árbol en una imagen que muestra una hilera de arboles y etiquetársela al árbol concreto del sistema), la ontología ya está preparada para ello, pero la API y la lógica subyacente no recogen esta funcionalidad, por lo que podría añadirse sin demasiada complejidad.

Del mismo modo, en el estado actual el sistema solo permite anotar la posición, especie y las imágenes de los árboles. Esta información es bastante básica y no es suficiente para usuarios que requieran datos más técnicos como medidas dendrométricas (incluidas en los datos del IFN). Añadir la posibilidad de anotar más características de los árboles como la altura, DAP... Supone la ampliación de la ontología creada (haciéndola más rica) y puede ampliar el abanico de usuarios de la aplicación, resultando interesante para ingenieros forestales o investigadores.

Respecto a la ontología, siguiendo la metodología de [16], podrían establecerse más restricciones sobre sus propiedades como la cardinalidad, características de las propiedades (indicar sin son funcionales, transitivas, simétricas/asimétricas, reflexivas/irreflexivas), definir las propiedades inversas de las existentes, clases disjuntas y restricciones avanzadas de OWL (de cuantificador, existenciales o universales) o mejorar la documentación de los términos definidos en la ontología, entre otras. Una ontología sistemática y precisa dota de más información a los datos anotados con los términos descritos en ella, por lo que son más reutilizables y útiles. Además, cuanto mejor anotados estén los datos y más completas las ontologías más interpretables son los datos por ordenadores, ya que disponen de más información.

En relación a la implementación de la API REST, aunque las pruebas de carga han demostrado un buen comportamiento del sistema, este se puede optimizar reduciendo las consultas al almacén de triplas. Esto es posible cacheando el área del territorio donde se encuentran los árboles. Para ello, una opción es definir zonas con forma de caja y almacenar en caché los puntos que definen estos polígonos. De esta forma, además de tener cacheada la información de los árboles, se conocería a priori si los árboles de una zona concreta están en memoria.

Finalmente, el hecho de que el sistema solo integre en el prototipo los árboles del IFN de la provincia de Valladolid, no es una limitación en sí misma. Del mismo modo que se ha realizado esta integración

de datos y con una consulta análoga a la utilizada (cambiando de provincia) se pueden exportar todos los datos publicados en Cross-Forest (referentes al territorio español) e importarlos al almacén de triplas propio. Por tanto, es inmediata la ampliación de los datos integrados del IFN a través del Cross-Forest. A largo plazo, se puede estudiar la posibilidad de extender esta información nacional a otros países de Europa o del mundo.

Referencias

[1] J. Breslin, A. Passant y S. Decker. *The Social Semantic Web*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1.ª edición, 2009.

[2] B. F. Lóscio, C. Burle y N. Calegari. Data on the Web Best Practices. W3C Recommendation, W3C, enero de 2017. https://www.w3.org/TR/2017/REC-dwbp-20170131/.

[3] CREAF. Ciencia ciudadana. Website. Último acceso julio 2020. http://www.creaf.cat/es/investigacion/ciencia-ciudadana.

[4] W. Hall y K. O'Hara. Semantic web. En *Computational Complexity*, páginas 2810-2829. Springer New York, 2012.

[5] W3C. Wiki, marzo de 2017. Use Cases/Social Semantic web. Último acceso julio 2020. https://www.w3.org/annotation/wiki/Use_Cases/Social_Semantic_web.

[6] B. Hyland, G. Atemezing, M. Pendleton y B. Srivastava. Linked Data Glossary. Working Group Note, W3C, junio de 2013. https://www.w3.org/TR/ld-glossary/.

[7] D. Allemang y J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann Publishers/Elsevier, Amsterdam; 2nd ed. Edición, 1 de julio de 2011.

[8] G. Vega, J. M. Giménez, C. Ordóñez y F. Bravo. Pioneering Easy-to-Use Forestry Data with Forest Explorer, abril de 2020. En revisión. Semantic Web Journal.

[9] Ministerio de Agricultura, pesca y alimentación. Gobierno de España. Inventario Forestal Nacional. Último acceso julio 2020. https://www.mapa.gob.es/es/desarrollo-rural/temas/politica-forestal/inventario-cartografia/inventario-forestal-nacional/default.aspx.

[10] Universidad de Valladolid, LinkedForest. Explorador Forestal. Website. Último acceso julio 2020. https://forestexplorer.gsic.uva.es/.

[11] Grupo de investigación GSIC/EMIC. Universidad de Valladolid. Website. Último acceso julio 2020. https://www.gsic.uva.es/.

[12] Cross-Forest. Website. Último acceso julio 2020. https://crossforest.eu/.

[13] R. Lasser. Electrical and Computer Engineering Design Handbook. Design Process. Engineering Method. Tufts University. 2013. Último acceso julio 2020. https://sites.tufts.edu/eeseniordesignhandbook/2013/engineering-method/.

[14] TreeTalk. Website. Último acceso julio 2020. https://www.treetalk.co.uk.

[15] PlantNet. Website. Último acceso julio 2020. https://plantnet.org/en/.

[16] N. Noy y D. Mcguinness. Ontology Development 101: A Guide to Creating Your First Ontology. *Knowledge Systems Laboratory*, 32, enero de 2001.

[17] I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, Boston, 9th ed. Edición.

[18] R. Cyganiak, D. Wood y M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Recommendation, W3C, febrero de 2015. https://www.w3.org/TR/rdf11-concepts/.

[19] D. Brickley y R. Guha. RDF Schema 1.1. Recommendation, W3C, febrero de 2015. https://www.w3.org/TR/rdf-schema/.

[20] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider y S. Rudolph. OWL 2 Web Ontology Language Primer (Second Edition). Working Recommendation, W3C, diciembre de 2012. https://www.w3.org/TR/owl-primer/.

[21] S. Harris y A. Seaborne. SPARQL 1.1 Query Language. Recommendation, W3C, marzo de 2013. https://www.w3.org/TR/sparql11-query/.

[22] Nginx. Admin Guide. Website. Último acceso julio 2020. https://docs.nginx.com/nginx/admin-guide/.

[23] Docker. Docker Docs. Website. Último acceso julio 2020. https://docs.docker.com/.

[24] Openstack Documentation. Launch and manage instances. Website. Último acceso julio 2020. https://docs.openstack.org/horizon/pike/user/launch-instances.html.

[25] Microsoft. Visual Studio Code. Website. Último acceso julio 2020. https://code.visualstudio.com/.

[26] Universidad de Standford. Protégé. Website. Último acceso julio 2020. https://protege.stanford.edu/.

[27] V. Link, S. Lohmann, E. Marbach, S. Negru y V. Wiens. WebVOWL. Website. Último acceso julio 2020. http://vowl.visualdataweb.org/webvowl.html.

[28] X. Jing. Ontology visualization, 2018. University of Southern California Information Sciences Institute. Repositorio GitHub. Último acceso agosto 2020. https://github.com/usc-isi-i2/ontology-visualization.

[29] Astah. Software Design Tools for Agile teams with UML, ER Diagram, Flowchart, Mindmap and More. Website. Último acceso julio 2020. http://astah.net/.

[30] Docker hub. nginx. Website. Último acceso julio 2020. https://hub.docker.com/_/nginx/.

[31] OpenLink Software. Virtuoso Open-Source Edition. Website. Último acceso julio 2020. http://vos.openlinksw.com/owiki/wiki/VOS.

[32] The LaTeXProject. Website. Último acceso mayo 2018. https://www.latex-project.org/.

[33] The MathWorks - MATLAB and Simulink for Technical Computing. Website. Último acceso septiembre 2020. http://www.mathworks.com/.

[34] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Informe técnico, University of California, Irvine, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[35] M. Duerst y M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987, RFC Editor, enero de 2005. http://www.rfc-editor.org/rfc/rfc3987.txt.

[36] J. Andrade. *Diseño y desarrollo de una aplicación web para el etiquetado socio-semántico en el ámbito de la gestión forestal*. Universidad de Valladolid. 2020.

[37] I. Jacobs y N. Walsh. Architecture of the World Wide Web, Volume One. Recommendation, W3C, diciembre de 2004. https://www.w3.org/TR/webarch/.

[38] T. Berners-Lee, R. T. Fielding y L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. STD 66, RFC Editor, enero de 2005. http://www.rfc-editor.org/rfc/rfc3986.txt.

[39] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach y T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, junio de 1999. http://www.rfc-editor.org/rfc/rfc2616.txt.

[40] DCMI Usage Board. DCMI Metadata Terms, enero de 2020. DCMI Recommendation. https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.

[41] F. Maali y J. Erickson. Data Catalog Vocabulary (DCAT). Informe técnico, W3C, enero de 2014. Recommendation. https://www.w3.org/TR/vocab-dcat/.

[42] T. Berners-Lee. Linked Data, julio de 2006. Website. Último acceso agosto 2020. https://www.w3.org/DesignIssues/LinkedData.html.

[43] M. Massé. *REST API Design Rulebook*. O'Reilly Media, Inc, USA, Gravenstein Highway North, Sebastopol, 1.ª edición, 11 de noviembre de 2011.

[44] B. Hyland, G. Atemezing y B. Villazón-Terrazas. Best Practices for Publishing Linked Data. Working Group Note, W3C, enero de 2017. https://www.w3.org/TR/ld-bp/.

[45] OECD. Open Government Data. Website. Último acceso julio 2020. http://www.oecd.org/internet/digital-government/open-government-data.htm.

[46] DBpedia. Website. Último acceso agosto 2020. https://wiki.dbpedia.org/.

[47] W3C. World Wide Web Consortium Supports the IETF URI Standard and IRI Proposed Standard. URI Specification Updated, IRIs Allow Internationalized Web Addressing. Informe técnico, noviembre de 2004. https://www.w3.org/2004/11/uri-iri-pressrelease.

[48] D. Brickley y L. Miller. FOAF Vocabulary Specification 0.99, enero de 2014. Namespace Document - Paddington Edition. http://xmlns.com/foaf/spec/.

[49] P. Bellini y P. Nesi. Performance assessment of RDF graph databases for smart city services. *Journal of Visual Languages & Computing*, 45:24-38, 2018.

[50] National Geospatial-Intelligence Agency (NGA). Department of Defense World Geodetic System 1984. Standardization Document. Versión 1.0.0, agosto de 2014. https://earth-info.nga.mil/GandG/publications/NGA_STND_0036_1_0_0_WGS84/NGA.STND.0036_1.0.0_WGS84.pdf.

[51] W3C Semantic Web Interest Group. *Basic Geo (WGS84 lat/long) Vocabulary*. Editado por D. Brickley. Website. Último acceso agosto 2020. https://www.w3.org/2003/01/geo/. W3C. Febrero de 2006.

[52] R. Troncy, J. van Ossenbruggen, J. Z. Pan y G. Stamou. Image Annotation on the Semantic Web. Incubator Group Report, W3C, agosto de 2007. https://www.w3.org/2005/Incubator/mmsem/XGR-image-annotation/.

[53] Ministerio de Agricultura, pesca y alimentación. Gobierno de España. Mapa Forestal de España. Último acceso julio 2020. https://www.mapa.gob.es/es/desarrollo-rural/temas/politica-forestal/inventario-cartografia/mapa-forestal-espana/default.aspx.

[54] J. M. Giménez. Data collection, Data quality, Ontologies and Vocabularies. Informe técnico. Versión 1.0, Cross-Forest Consortium, noviembre de 2019. Entregable D2.1.

[55] Esri. ArcGIS for Desktop. Qué es un shapefile. Website. Último acceso septiembre 2020. https://desktop.arcgis.com/es/arcmap/10.3/manage-data/shapefiles/what-is-a-shapefile.htm.

[56] B. Fierro, T. Jurado, J. M. Giménez y G. Vega. Data exportation and Publication - Interim Report. Informe técnico. Versión 1.0, Cross-Forest Consortium, febrero de 2020. Entregable D2.2.

[57] R. Stephens. *Beginning Software Engineering*. John Wiley & Sons Inc, Indianapolis, Indiana, marzo de 2015.

[58] Real Academia Española. Diccionario de la lengua española. Website. Último acceso septiembre 2020. https://dle.rae.es/.

[59] A. Freier, P. Karlton y P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101, RFC Editor, agosto de 2011. http://www.rfc-editor.org/rfc/rfc6101.txt.

[60] K. Dubost. *Exif vocabulary workspace - RDF Schema*. Website. Último acceso agosto 2020. https://www.w3.org/2003/12/exif/. W3C. Febrero de 2014.

[61] GeoRepository. EPSG Geodetic Parameter Dataset. Website. Último acceso septiembre 2020. https://epsg.org/home.html.

[62] Linked Open Vocabularies (LOV). Website. Último acceso agosto 2020. https://lov.linkeddata.es/dataset/lov.

[63] Schema.org. Website. Último acceso agosto 2020. https://schema.org/.

[64] W3C. Wiki, febrero de 2005. Image Description Rdf Examples. Website. Último acceso agosto 2020. https://www.w3.org/wiki/ImageDescriptionRdfExamples.

[65] P. Nair, M. Moh y T. Moh. Using social media presence for alleviating cold start problems in privacy protection. En *2016 International Conference on Collaboration Technologies and Systems (CTS)*, páginas 11-17, 2016.

[66] K. Page, D. De Roure y K. Martinez. REST and Linked Data: A match made for domain driven development?, marzo de 2011.

[67] OpensJS Foundation. Node.js Documentation. Website. Último acceso julio 2020. https://nodejs.org/es/docs/.

[68] OpensJS Foundation. Express. Website. Último acceso julio 2020. https://expressjs.com/es/.

[69] npm. Website. Último acceso agosto 2020. https://www.npmjs.com/.

[70] J. Andrade y C. Mayo. Social Tree Annotation, 2020. GSIC/EMIC. Universidad de Valladolid. Repositorio GitHub. Último acceso septiembre 2020. `https://github.com/gsic-emic/SocialTreeAnnotation`.

[71] tenforce. Virtuoso docker, 2019. Website. Último acceso agosto 2020. `https://hub.docker.com/r/tenforce/virtuoso`.

[72] OpenLink Software. Virtuoso Open-Source Edition, agosto de 2018. Website. Último acceso agosto 2020. `https://github.com/openlink/virtuoso-opensource`.

[73] Y. Singh. *Software Testing*. Cambridge University Press, Cambridge, 2009.

[74] cURL. Command line tool and library for transferring data with URLs. Website. Último acceso septiembre 2020. `https://curl.haxx.se/`.

[75] P. Rottmann. APIDOC, Inline Documentation for RESTful web APIs. Website. Último acceso agosto 2020. `https://apidocjs.com/`.

Apéndice ***A***

Ontología STA (*Social Tree Annotation*)

```
@prefix sta: <http://timber.gsic.uva.es/sta/ontology/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ifn: <http://crossforest.eu/ifn/ontology/> .
@prefix vann: <http://purl.org/vocab/vann/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

sta: a owl:Ontology ;
    rdfs:label "Social Tree Annotation Ontology"@en , "Ontología de anotación social de
    árboles"@es ;
    rdfs:comment "Ontology to publish data of the Social Forest inventory"@en , "Ontología para
    publicar datos del Inventario Forestal Social" ;
    owl:versionInfo 0.1 ;
    dc:creator <https://www.gsic.uva.es/members/cmayo> ;
    dc:date "2019-10-12"^^xsd:date ;
    owl:imports sta: , ifn: ;
    vann:preferredNamespacePrefix "sta" . #The preferred namespace prefix to use when using
    terms  from this vocabulary in an XML document.

#################################################################
#    Classes
#################################################################

sta:Tree a owl:Class , rdfs:Class ;
    rdfs:label "Tree"@en , "Árbol"@es ;
    rdfs:comment "Woody perennial plant with an elongated stem supporting
    branches"@en ,  "Planta vivaz que vive más de dos años, de tallo leñoso,
    que se ramifica a cierta altura del suelo"@es ;
    rdfs:isDefinedBy sta: .

sta:TreePartPhoto a owl:Class , rdfs:Class ;
    rdfs:label "Tree part photo"@en , "Parte del árbol de la foto"@es ;
    rdfs:comment "Part of a tree that it is shown in the image"@en , "Parte del
    árbol que se muestra en la imagen"@es ;
    rdfs:isDefinedBy sta: .

sta:LeafPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
```

```
    rdfs:label "Leaf"@en , "Hoja"@es ;
    rdfs:comment "Leaf"@en , "Hoja de árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:FruitPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "Fruit"@en , "Fruto"@es ;
    rdfs:comment "Fruit which grows on a tree"@en , "Fruto que crece en un árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:TrunkPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "Trunk"@en , "Tronco"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:isDefinedBy sta: .

sta:BranchPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "Branch"@en , "Rama"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:isDefinedBy sta: .

sta:CrownPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "Crown"@en , "Copa"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:isDefinedBy sta: .

sta:GeneralViewPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "General"@en , "General"@es ;
    rdfs:comment "General view"@en , "Vista general"@es ;
    rdfs:isDefinedBy sta: .

sta:FlowerPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "Flower"@en , "Flor"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:isDefinedBy sta: .

sta:OtherPartPhoto a owl:Class , rdfs:Class ;
    rdfs:subClassOf sta:TreePartPhoto ;
    rdfs:label "Other part"@en , "Otra parte"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:isDefinedBy sta: .

sta:Image a owl:Class , rdfs:Class ;
    rdfs:subClassOf foaf:Image ;
    rdfs:label "Image"@en , "Imagen"@es ;
    rdfs:comment "Tree's image"@en , "Imagen de un árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:Annotation a owl:Class, rdfs:Class ;
    rdfs:label "Annotation"@en , "Anotación"@es ;
    rdfs:comment "Tree's annotation"@en , "Anotación de un árbol"@es ;
```

```
    rdfs:isDefinedBy sta: .

sta:SpeciesAnnotation a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:Annotation ;
    rdfs:label "Species annotation"@en , "Anotación de especie"@es ;
    rdfs:comment "Annotation about a tree species"@en , "Anotación sobre la
    especie de un árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:PositionAnnotation a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:Annotation ;
    rdfs:label "Position annotation"@en , "Anotación de posición"@es ;
    rdfs:comment "Annotation about a tree position"@en , "Anotación sobre la
    posición de un árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:ImageAnnotation a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:Annotation ;
    rdfs:label "Image annotation"@en , "Anotación de imagen"@es ;
    rdfs:comment "Annotation about a tree image"@en , "Anotación sobre la imagen de un árbol"@es
    ;
    rdfs:isDefinedBy sta: .

#La anotación primaria es la más votada en un momento dado (los datos que se consideran
"válidos")
sta:PrimaryAnnotation a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:Annotation ;
    rdfs:label "Primary annotation"@en , "Anotación primaria"@es ;
    rdfs:comment "Primary annotation about a tree, which is considered the most popular at a
    given time"@en , "Anotación
    primaria sobre un árbol, considerada la más votada en un momento dado"@es ;
    rdfs:isDefinedBy sta: .

sta:PrimaryPosition a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:PrimaryAnnotation ;
    rdfs:label "Primary position"@en , "Posición primaria"@es ;
    rdfs:comment "Primary position annotation about a tree, which is considered the most
    popular at a given time"@en , "Anotación primaria de posición sobre un árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:PrimarySpecies a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:PrimaryAnnotation ;
    rdfs:label "Primary species"@en , "Especie primaria"@es ;
    rdfs:comment "Primary species annotation about a tree"@en , "Anotación
    primaria de especie sobre un árbol, considerada
    la más votada en un momento dado"@es ;
    rdfs:isDefinedBy sta: .

sta:AssertedAnnotation a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:PrimaryAnnotation ;
    rdfs:label "Asserted annotation"@en , "Anotación afirmada"@es ;
    rdfs:comment "Annotation asserted by experts about a tree"@en , "Anotación
    afirmada por expertos sobre un árbol"@es ;
    rdfs:isDefinedBy sta: .
```

```
sta:AssertedPosition a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:AssertedAnnotation ;
    rdfs:label "Asserted position"@en , "Posición afirmada"@es ;
    rdfs:comment "Annotation asserted by experts about a tree position"@en ,
    "Anotación afirmada por expertos sobre una posición de un árbol"@es ;
    rdfs:isDefinedBy sta: .

sta:AssertedSpecies a owl:Class, rdfs:Class ;
    rdfs:subClassOf sta:AssertedAnnotation ;
    rdfs:label "Asserted species"@en , "Especie afirmada"@es ;
    rdfs:comment "Annotation asserted by experts about a tree species"@en ,
    "Anotación afirmada por expertos sobre una especie de un árbol"@es ;
    rdfs:isDefinedBy sta: .

#################################################################
#    Object Properties
#################################################################
sta:hasAnnotation a owl:ObjectProperty , rdf:Property ;
    rdfs:label "has annotation"@en , "tiene anotación"@es ;
    rdfs:comment "annotation of a tree"@en , "anotación de un árbol"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:Annotation ;
    rdfs:isDefinedBy sta: .

sta:hasSpeciesAnnotation a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasAnnotation ;
    rdfs:label "has species annotation"@en , "tiene anotación de especie"@es ;
    rdfs:comment "annotation of species"@en , "anotación de una especie"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:SpeciesAnnotation ;
    rdfs:isDefinedBy sta: .

sta:hasPositionAnnotation a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasAnnotation ;
    rdfs:label "has position annotation"@en , "tiene anotación de posición"@es ;
    rdfs:comment "annotation of a tree position"@en , "anotación de una posición de un árbol"@es
    ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:PositionAnnotation ;
    rdfs:isDefinedBy sta: .

sta:hasImageAnnotation a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasAnnotation ;
    rdfs:label "has image annotation"@en , "tiene anotación de imagen"@es ;
    rdfs:comment "annotation which includes a image"@en , "anotación que incluye una imagen"@es
    ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:ImageAnnotation ;
    rdfs:isDefinedBy sta: .

sta:hasImage a owl:ObjectProperty , rdf:Property ;
    rdfs:label "has image"@en , "tiene imagen"@es ;
    rdfs:comment "image of a tree or a part of it"@en , "imagen de un árbol o de una parte"@es ;
    rdfs:domain sta:ImageAnnotation ;
    rdfs:range sta:Image ;
```

```
    rdfs:isDefinedBy sta: .

# Propiedad más general que especie (no es subpropiedad del ifn:hasTaxon porque
tiene distinto dominio)
sta:hasTaxon a owl:ObjectProperty , rdf:Property ;
    rdfs:label "has taxon"@en , "tiene taxón"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:domain sta:SpeciesAnnotation ;
    rdfs:range ifn:Taxon ;
    rdfs:isDefinedBy sta: .

sta:hasPrimaryAnnotation a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasAnnotation ;
    rdfs:label "has primary annotation"@en , "tiene anotación primaria"@es ;
    rdfs:comment "primary annotation of a tree"@en , "anotación primaria de un árbol"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:PrimaryAnnotation ;
    rdfs:isDefinedBy sta: .

# Propiedad proxy, es la posición que se considera buena para un árbol en cada momento
sta:hasPrimaryPosition a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasPrimaryAnnotation ;
    rdfs:label "has primary position annotation"@en , "tiene anotación primaria de posición"@es
    ;
    rdfs:comment "primary annotation of a tree position"@en , "anotación primaria de una
    posición de un árbol"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:PrimaryPosition ;
    rdfs:isDefinedBy sta: .

sta:hasPrimarySpecies a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasPrimaryAnnotation ;
    rdfs:label "has primary species annotation"@en , "tiene anotación primaria
    de especie"@es ;
    rdfs:comment "primary annotation of a tree species"@en , "anotación primaria de una especie
    de un árbol"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:PrimarySpecies ;
    rdfs:isDefinedBy sta: .

sta:hasAssertedAnnotation a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasPrimaryAnnotation ;
    rdfs:label "has asserted annotation"@en , "tiene anotación afirmada"@es ;
    rdfs:comment "annotation asserted by experts of a tree"@en , "anotación
    afirmada por expertos de un árbol"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:AssertedAnnotation ;
    rdfs:isDefinedBy sta: .

sta:hasAssertedPosition a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasAssertedAnnotation ;
    rdfs:label "has asserted position"@en , "tiene posición afirmada"@es ;
    rdfs:comment "annotation asserted by experts of a tree position"@en ,
    "anotación afirmada por expertos de una posición un árbol"@es ;
    rdfs:domain sta:Tree ;
```

```
    rdfs:range sta:AssertedPosition ;
    rdfs:isDefinedBy sta: .

sta:hasAssertedSpecies a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf sta:hasAssertedAnnotation ;
    rdfs:label "has asserted species"@en , "tiene especie afirmada"@es ;
    rdfs:comment "annotation asserted by experts of a tree species"@en ,
    "anotación afirmada por expertos de una especie de un árbol"@es ;
    rdfs:domain sta:Tree ;
    rdfs:range sta:AssertedSpecies ;
    rdfs:isDefinedBy sta: .

# Para enlazar el jpg con la imagen
sta:resource a owl:ObjectProperty , rdf:Property ;
    rdfs:subPropertyOf foaf:depiction ;
    rdfs:label "resource"@en , "recurso"@es ;
    rdfs:comment ""@en , ""@es ;
    rdfs:domain sta:Image ;
    rdfs:isDefinedBy sta: .
```

Apéndice **B**

Documentación API REST

La documentación de la API REST presentada en este apéndice ha sido generada con apiDoc [75].

API-REST-SocialTreeApp-Timber

API REST para aplicación socio-semántica de anotación de árboles

Anotación

Anotación - Consultar anotaciones del sistema

GET

```
/data/annotation
```

query

Campo	Tipo	Descripción
page	Number	Número de página para resultados paginados
creator	String	Nombre de usuario para filtrar árboles creados por un usuario en el sistema

Success 200

Campo	Tipo	Descripción
response	Object	Anotaciones registradas en el sistema filtradas por creador

204 No Content

Campo	Tipo	Descripción
null	null	No hay árboles en el sistema

400 BadRequest

Nombre	Descripción
BadRequest	Petición errónea
userNoExist	Es necesario indicar un creador

500 Internal Server Error

Nombre	Descripción
conexionVirtuoso	Error de conexión con el endpoint SPARQL

Anotación - Consultar información de una anotación

GET

```
/data/annotation/:annotationId
```

path

Campo	Tipo	Descripción

Campo	Tipo	Descripción
`anotationId`	String	Identificador de anotación

Success 200

Campo	Tipo	Descripción
`response`	Object	Información de una anotación del sistema

404 Not Found

Nombre	Descripción
`annotationNotFound`	El `anotationId` no se ha encontrado

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Anotación - Crear nueva anotación

POST

```
/data/annotation
```

Header

Campo	Tipo	Descripción
`Authorization`	String	Autenticación básica HTTP

body

Campo	Tipo	Descripción
`id`	String	URI completa del árbol al que se va a asociar (obligatorio)
`type`	String	Tipo de anotación: postion, species, image (obligatorio)
`creator`	String	URI completa del usuario creador (obligatorio)
`lat`	Number	Coordenada de longitud para la posición del árbol (obligatorio en anotación de tipo posición)
`long`	Number	Coordenada de longitud para la posición del árbol (obligatorio en anotación de tipo posición)
`species`	String	Especie de la taxonomía (obligatorio en anotación de tipo especie)
`image`	String	Imagen del árbol codificada en base64 (obligatorio en anotación de tipo imagen)
`title`	String	Título de la imagen subida (optativo en anotación de tipo imagen)
`description`	String	Descripción de la imagen subida(optativo en anotación de tipo imagen)

Campo	Tipo	Descripción
depicts	String	URI completa del elemento que muestra la imagen subida, es una de las opciones definidas en la ontología (subclases de TreePartPhoto) (optativo en anotación de tipo imagen)

201 Created

Campo	Descripción
annotCreated	Anotación creada correctamente

401 Unathorized

Nombre	Descripción
Unathorized	Login/Password incorrecto

400 BadRequest

Nombre	Descripción
emptyRequiredFields	Faltan campos obligatorios
treeNoExist	El árbol no existe
errorExif	Error leyendo exif imagen

500 Internal Server Error

Nombre	Descripción
conexionVirtuoso	Error de conexión con el endpoint SPARQL
internalError	Error interno
errorCache	Error cacheando objeto

Anotación - Eliminar árbol

No implementado

DELETE

```
/data/annotation/:annotationId
```

Anotación - Modificar anotación

No implementado

PUT

```
/data/annotation/:annotationId
```

Especie

Especie - Consultar especies del sistema

GET

```
/data/species
```

query

Campo	Tipo	Descripción
page	Number	Número de página para resultados paginados

Success 200

Campo	Tipo	Descripción
response	Object	Especies registrados en el sistema (provenientes de la taxonomía del Cross-Forest)

500 Internal Server Error

Nombre	Descripción
conexionVirtuoso	Error de conexión con el endpoint SPARQL

Imagen

Imagen - Listar imágenes

No implementado (se sirve com contenido estático fuera de la API)

GET

```
/data/image
```

Imagen - Consultar información de una imagen o recuperar un jpg

GET

```
/data/image/:imageId
```

path

Campo	Tipo	Descripción
imageId	String	Identificador de imagen

Success 200

Campo	Tipo	Descripción
response	Object	Información de una imagen del sistema

404 Not Found

Nombre	Descripción
`TreeNotFound`	El `imageId` no se ha encontrado

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Partes de Árbol

Partes de Árbol - Consultar partes del árbol para etiquetar imágenes

GET

```
/data/tree
```

Success 200

Campo	Tipo	Descripción
`response`	Object	Partes de árbol para etiquetar en una imagen definidas en la ontología

204 No Content

Campo	Tipo	Descripción
`null`	null	No hay árboles en el sistema

400 BadRequest

Nombre	Descripción
`BadRequest`	Petición errónea

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Raíz

Raíz - Iniciar sesión

POST

```
/
```

body

Campo	Tipo	Descripción
`idUser`	String	Nombre de usuario
`password`	String	Contraseña de usuario

Success 200

Campo	Tipo	Descripción
`response`	Object	Autorizado

401 Unathorized

Nombre	Descripción
`Unathorized`	Login/Password incorrecto

Raíz - Inicio

GET

```
/
```

Success 200

Campo	Tipo	Descripción
`response`	Object	Urls definidas en la API REST

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Usuario

Usuario - Consultar información de un usuario

GET

```
/data/user/:userId
```

path

Campo	Tipo	Descripción
`userId`	String	Identificador de usuario

Success 200

Campo	Tipo	Descripción
`response`	Object	Información de un usuario del sistema

404 Not Found

Nombre	Descripción
`UserNotFound`	El `userId` no se ha encontrado

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Usuario - Consultar usuarios del sistema

GET

```
/data/user
```

query

Campo	Tipo	Descripción
`page`	Number	Número de página para resultados paginados

Success 200

Campo	Tipo	Descripción
`response`	Object	Usuarios registrados en el sistema

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Usuario - Crear/Actualizar usuario registrado

Actualización de información de usuario registrado no implementado

PUT

```
/data/user/:userId
```

Header

Campo	Tipo	Descripción
`Authorization`	String	Autenticación básica HTTP

body

Campo	Tipo	Descripción
`nombre`	String	Nombre y apellidos (obligatorio)
`email`	String	Correo electrónico (obligatorio)
`password`	String	Contraseña de usuario (obligatorio)

path

Campo	Tipo	Descripción
`userId`	String	Identificador de usuario (nombre de usuario)

201 Created

Campo	Descripción
`userCreated`	Usuario creado correctamente

401 Unathorized

Nombre	Descripción
`Unathorized`	Login/Password incorrecto

400 BadRequest

Nombre	Descripción
`emptyRequiredFields`	Faltan campos obligatorios
`usedLogin`	Login ya usado

404 Not Found

Nombre	Descripción
`UserNotFound`	El `userId` no se ha encontrado

405 Method Not Allowed

Nombre	Descripción
`MethodNotAllowed`	Método no permitido para el recurso (No se implementa la actualización)

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL
`hashPasswd`	Error creando hash de la contraseña

Usuario - Eliminar usuario

No implementado

DELETE

```
/data/user/:userId
```

Árbol

Árbol - Consultar información de un árbol

GET

```
/data/tree/:treeId
```

path

Campo	Tipo	Descripción
`treeId`	String	Identificador de árbol

Success 200

Campo	Tipo	Descripción
`response`	Object	Información de un árbol del sistema

404 Not Found

Nombre	Descripción
`TreeNotFound`	El `treeId` no se ha encontrado

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL

Árbol - Consultar árboles del sistema

GET

```
/data/tree
```

query

Campo	Tipo	Descripción
`page`	Number	Número de página para resultados paginados
`lat0`	Number	Latitud 0 para filtrar árboles en un área
`lat1`	Number	Latitud 1 para filtrar árboles en un área
`long0`	Number	Longitud 0 para filtrar árboles en un área
`long1`	Number	Longitud 1 para filtrar árboles en un área
`species`	String	Especie para filtrar árboles del sistema
`creator`	String	Nombre de usuario para filtrar árboles creados por un usuario en el sistema

Success 200

Campo	Tipo	Descripción

Campo	Tipo	Descripción
response	Object	Árboles registrados en el sistema (todos, filtrados en un área, filtrados por especie, filtrados por creador)

204 No Content

Campo	Tipo	Descripción
null	null	No hay árboles en el sistema

400 BadRequest

Nombre	Descripción
BadRequest	Petición errónea

500 Internal Server Error

Nombre	Descripción
conexionVirtuoso	Error de conexión con el endpoint SPARQL

Árbol - Crear nuevo árbol

POST

```
/data/tree
```

Header

Campo	Tipo	Descripción
Authorization	String	Autenticación básica HTTP

body

Campo	Tipo	Descripción
creator	String	URI completa del usuario creador (obligatorio)
lat	Number	Coordenada de latitud para la posición del árbol (obligatorio)
long	Number	Coordenada de longitud para la posición del árbol (obligatorio)
image	String	Imagen del árbol codificada en base64
species	String	Especie de la taxonomía del Cross-Forest
title	String	Título de la imagen subida
description	String	Descripción de la imagen subida
depicts	String	URI completa del elemento que muestra la imagen subida, es una de las opciones definidas en la ontología (subclases de TreePartPhoto)

201 Created

Campo	Descripción
`treeCreated`	Árbol creado correctamente

401 Unathorized

Nombre	Descripción
`Unathorized`	Login/Password incorrecto

400 BadRequest

Nombre	Descripción
`emptyRequiredFields`	Faltan campos obligatorios

500 Internal Server Error

Nombre	Descripción
`conexionVirtuoso`	Error de conexión con el endpoint SPARQL
`errorCache`	Error cacheando objeto

Árbol - Eliminar árbol

No implementado

DELETE

```
/data/tree/:treeId
```

Árbol - Modificar árbol

No implementado

PUT

```
/data/tree/:treeId
```

Apéndice **C**

Guía para OpenLink Virtuoso OpenSource Edition 7.2.5

Guía para la instalación, configuración y uso de Virtuoso en Linux 64 bits [31, 72]:

1. Instalación de dependencias:

```
$ sudo apt-get install autoconf automake libtool flex bison gperf gawk m4 make
$ sudo apt-get install git wget curl libreadline-dev openssl libssl-dev
```

   Instalación y compilación de la versión 1.0.2p de OpenSSL, ya que la instalación de Virtuoso falla para versiones superiores:

```
$ sudo apt install build-essential checkinstall zlib1g-dev -y
$ cd /usr/local/src/
$ sudo wget https://www.openssl.org/source/openssl-1.0.2o.tar.gz
$ sudo tar -xf openssl-1.0.2o.tar.gz
$ cd openssl-1.0.2o
$ cd /usr/local/src/openssl-1.0.2o
$ sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib
$ sudo make
$ sudo make test
$ sudo make install
$ cd /etc/ld.so.conf.d/
$ sudo vim openssl-1.0.2o.conf
            /usr/local/ssl/lib
$ sudo ldconfig -v
$ sudo mv /usr/bin/c_rehash /usr/bin/c_rehash.BEKUP
$ sudo mv /usr/bin/openssl /usr/bin/openssl.BEKUP
$ sudo vim /etc/environment
            :/usr/local/ssl/bin
$ source /etc/environment
$ openssl version -a
```

2. Instalación y compilación de Virtuoso:

```
$ sudo git clone https://github.com/openlink/virtuoso-opensource.git /tmp/virtuoso
# sudo ./autogen.sh (cerciorarse que está el openssl en $PATH de root)
# export CFLAGS="-O2 -m64" (para Linux 64bits)
# ./configure --enable-openssl=/usr/local/ssl/
```

```
# make
# make install
```

Tras completar esta fase, el Virtuoso está instalado en `/usr/local/virtuoso-opensource`.

3. Reinstalar la versión actualizada de OpenSSL deshaciendo los pasos anteriores para eliminar los ficheros de la versión antigua.

4. Configuración: se encuentra en el fichero `/usr/local/virtuoso-opensource/var/lib/virtuoso/db/virtuoso.ini`. Teniendo en cuenta que el servidor tiene 4GB de memoria RAM y tendrá aproximadamente 2GB libres se establece la siguiente configuración:

```
NumberOfBuffers = 170000
MaxDirtyBuffers = 130000
```

5. Ejecución:

```
$ cd /usr/local/virtuoso-opensource/var/lib/virtuoso/db/
# sudo /usr/local/bin/virtuoso-t -f &
```

6. Acceso a consola con usuario `dba`:

```
# /usr/local/bin/isql 1111 dba dba (Pass por defecto dba)
```

Se puede modificar la contraseña por defecto del usuario:

```
SQL > set password "dba" XXXXXX;
```

7. Carga de datos en ficheros Turtle: se copian los ficheros *.ttl* a la ruta `/usr/local/virtuoso-opensource/var/lib/virtuoso/db/` y se ejecutan los comandos siguientes sustituyendo los valores entre "<>" por los adecuados:

```
SQL > ld_dir('<ruta_virtuoso>', '<fichero_datos.ttl>', '<grafo>');
SQL > rdf_loader_run();
SQL > select * from DB.DBA.load_list
SQL > checkpoint;
```

8. Borrado de datos cargados:

```
SQL> RDF_GLOBAL_RESET ();
SQL> delete from DB.DBA.load_list where ll_file='<./fichero_datos.ttl>' ;
```

9. Parada de Virtuoso:

```
SQL> shutdown;
```

Apéndice ***D***

Ficheros unit de configuración de servicios de systemd

Fichero *unit* de configuración de servicio de `systemd` con extensión *.service* para la API REST [68]:

```
[Unit]
Description=Backend Timber App

[Service]
Type=simple
ExecStart=/usr/local/bin/node $HOME/SocialTreeAnnotation/Cristina$/service_REST_timber/app.js
WorkingDirectory=/home/ubuntu/SocialTreeAnnotation/Cristina/service_REST_timber

User=ubuntu
Group=ubuntu

# Environment variables:
Environment=NODE_ENV=production

# Allow many incoming connections
LimitNOFILE=infinity

# Allow core dumps for debugging
LimitCORE=infinity

StandardInput=null
StandardOutput=file:/var/log/timber/nodejs.log
StandardError=file:/var/log/timber/err.log
Restart=always
#RestartSec=10
[Install]
WantedBy=multi-user.target
```

Fichero *unit* de configuración de servicio de `systemd` con extensión *.service* para OpenLink Virtuoso:

```
[Unit]
Description=Backend Timber App: Virtuoso

[Service]
Type=simple
ExecStart=/usr/local/bin/virtuoso-t -f
```

```
WorkingDirectory=/usr/local/virtuoso-opensource/var/lib/virtuoso/db/
Restart=always
[Install]
WantedBy=multi-user.target
```

Apéndice ***E***

Configuración de Nginx

Fichero de configuración de *host* virtuales de Nginx completo:

```
# Configuración de la caché
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m max_size=4g
    inactive=600m use_temp_path=off;

# VirtualHost para HTTPS
server {
    listen       443 ssl;
    server_name  timber.gsic.uva.es;
    ssl_certificate /etc/nginx/certs/gsic.uva.es.crt;
    ssl_certificate_key /etc/nginx/certs/gsic.uva.es.key;

    # Permito cuerpo de tamaño hasta 15MB para las imágenes que van en el json
    client_max_body_size 15M;

    # Configuración compresión gzip
    gzip on;
    gzip_types      text/plain application/xml application/sparql-results+json
                    text/javascript text/js text/css application/javascript
                    application/json application/x-javascript image/jpeg;
    gzip_proxied    no-cache no-store private expired auth;
    gzip_min_length 100;

    # Cabecera de estado de la caché
    add_header X-Cache-Status $upstream_cache_status;

    charset UTF-8;
    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    root /usr/share/nginx/html;

    # Se puede configurar un front-end en la ruta raíz
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html =404;
    }

    # Servicio web Nodejs => Backend
```

```
    location /sta {
        proxy_buffers 16 4k;
        proxy_buffer_size 2k;
        proxy_pass http://timber.gsic.uva.es:8888;
    }

    # Punto SPARQL
    location /sparql {
        proxy_cache my_cache;
        proxy_cache_key "$request_uri";
        proxy_ignore_headers Cache-Control;
        #proxy_cache_valid any 300m;
        proxy_pass http://timber.gsic.uva.es:8890;
    }

    # Imágenes
    location /data/images/ {
        autoindex on;
        sendfile on; # Evita copiar los datos en un buffer
        tcp_nopush on;
        proxy_cache my_cache;
        proxy_cache_key "$request_uri";
        proxy_ignore_headers Cache-Control;
    }
    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}

# VirtualHost para HTTP
server {
    listen 80;
    server_name  timber.gsic.uva.es;
    return 301 https://$server_name$request_uri;
}
```

Apéndice ***F***

Boxplots de tiempos de respuesta del sistema

Boxplots para los experimentos de las pruebas de carga definidas en el capítulo 6.

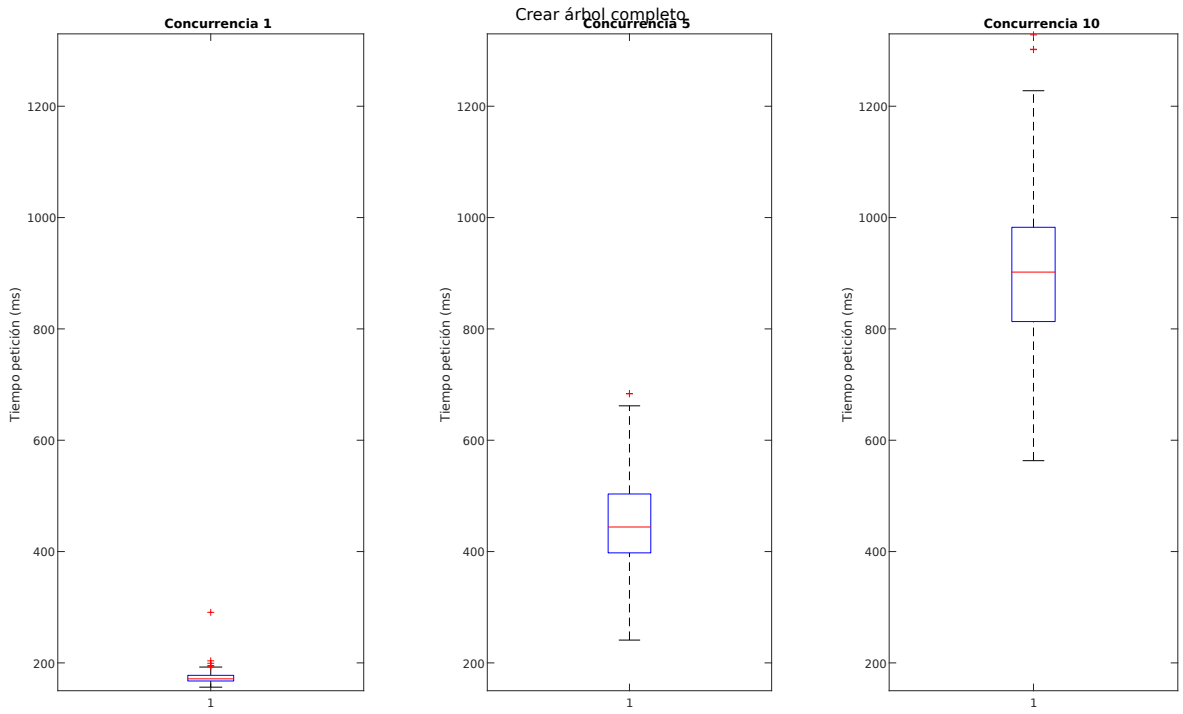


Figura F.1: *Boxplots* de los tiempos de respuesta para peticiones de crear árbol con `loadtest` para 30 segundos de experimento.

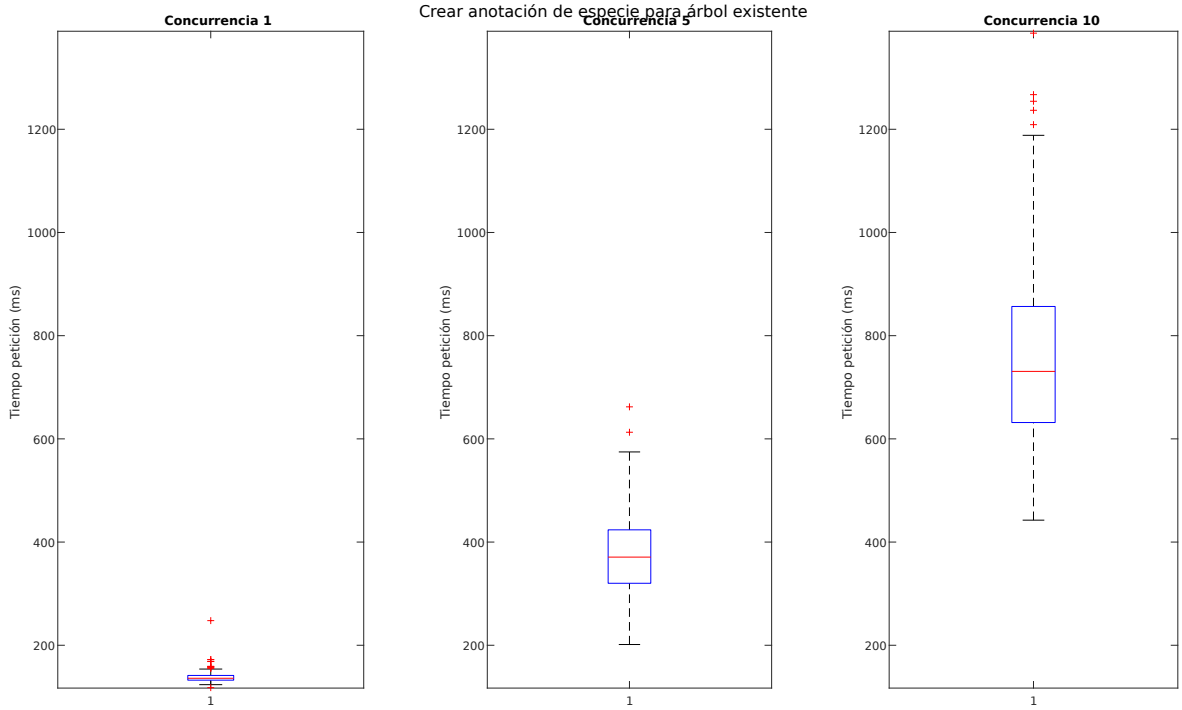


Figura F.2: *Boxplots* de los tiempos de respuesta para peticiones de crear anotación de especie para un árbol con `loadtest` para 30 segundos de experimento.

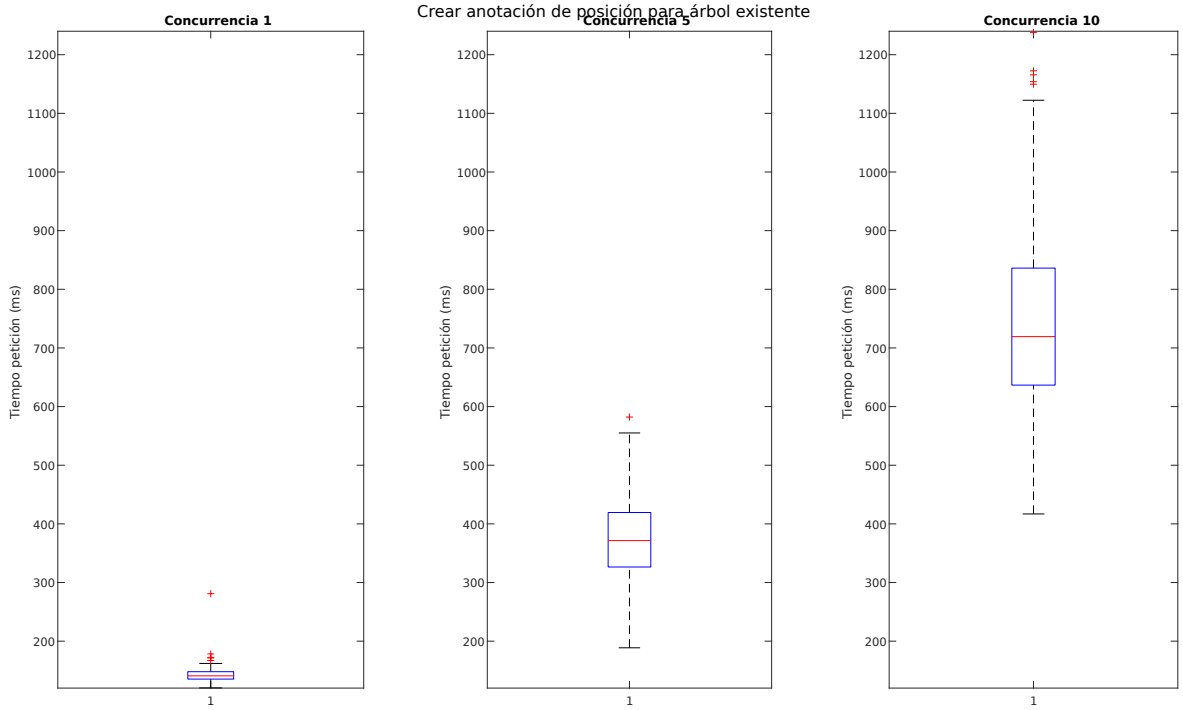


Figura F.3: *Boxplots* de los tiempos de respuesta para peticiones de crear anotación de posición para un árbol con `loadtest` para 30 segundos de experimento.

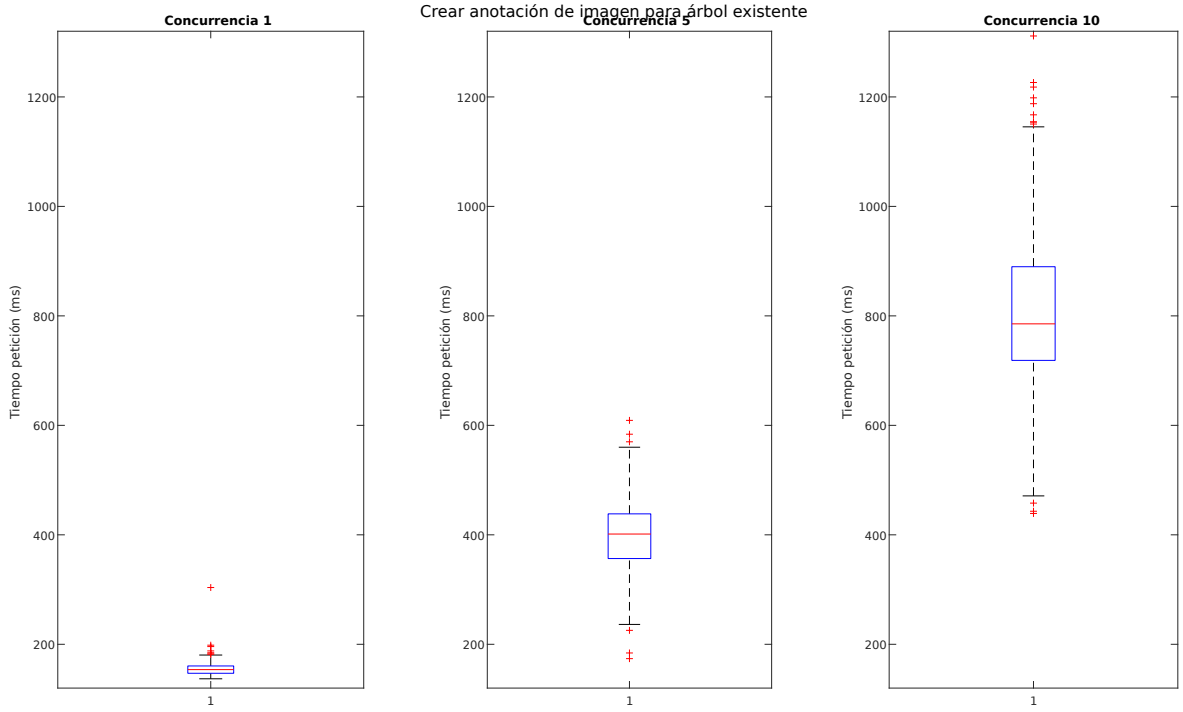


Figura F.4: *Boxplots* de los tiempos de respuesta para peticiones de crear anotación de imagen para un árbol con `loadtest` para 30 segundos de experimento.

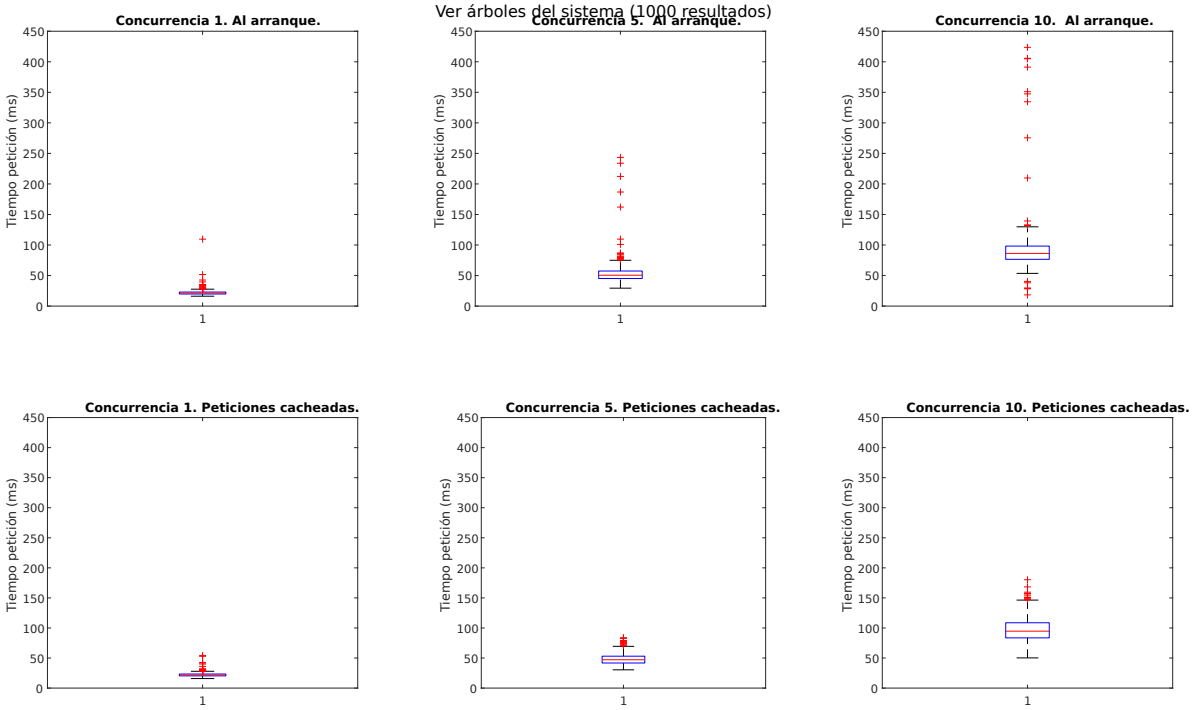


Figura F.5: *Boxplots* de los tiempos de respuesta para peticiones de ver árboles con `loadtest` para 30 segundos de experimento.

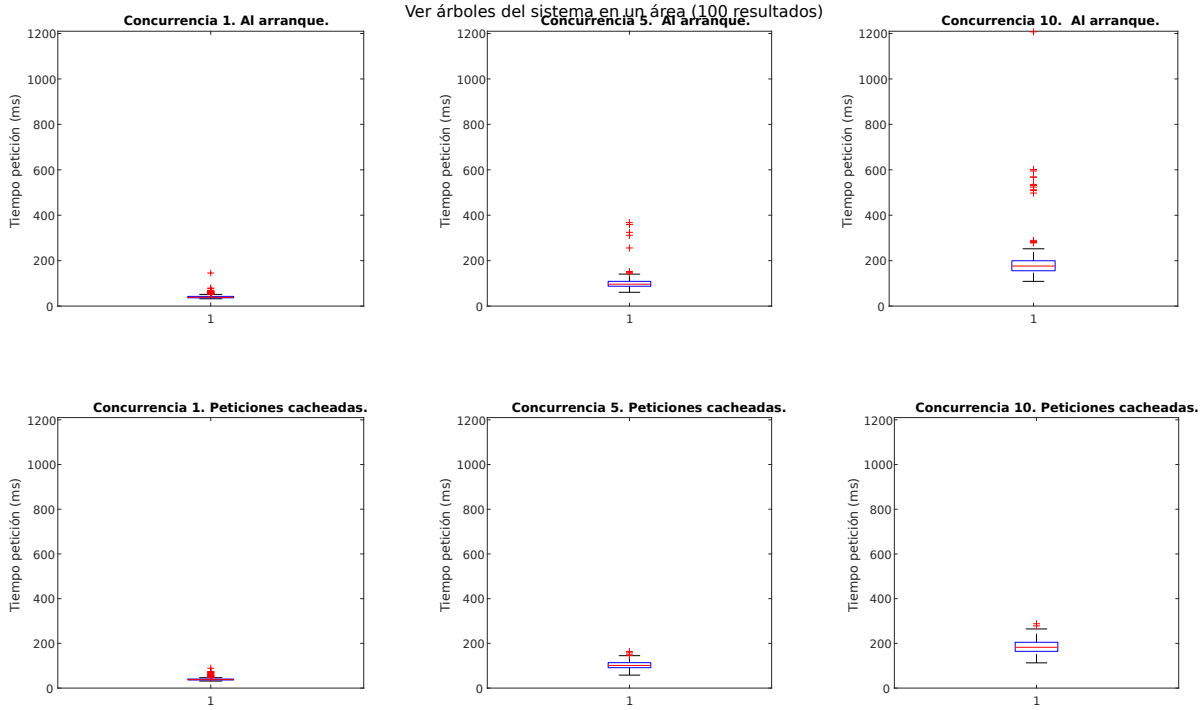


Figura F.6: *Boxplots* de los tiempos de respuesta para peticiones de ver árboles en un área con `loadtest` para 30 segundos de experimento.

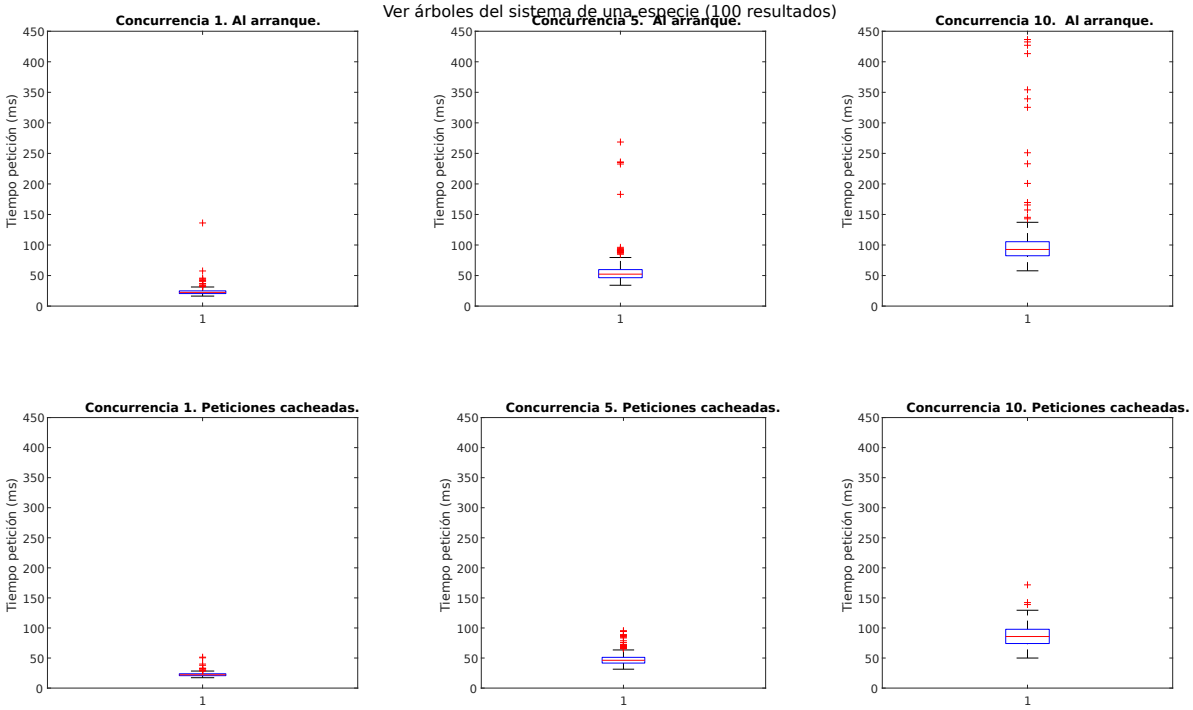


Figura F.7: *Boxplots* de los tiempos de respuesta para peticiones de ver árboles de una especie con `loadtest` para 30 segundos de experimento.

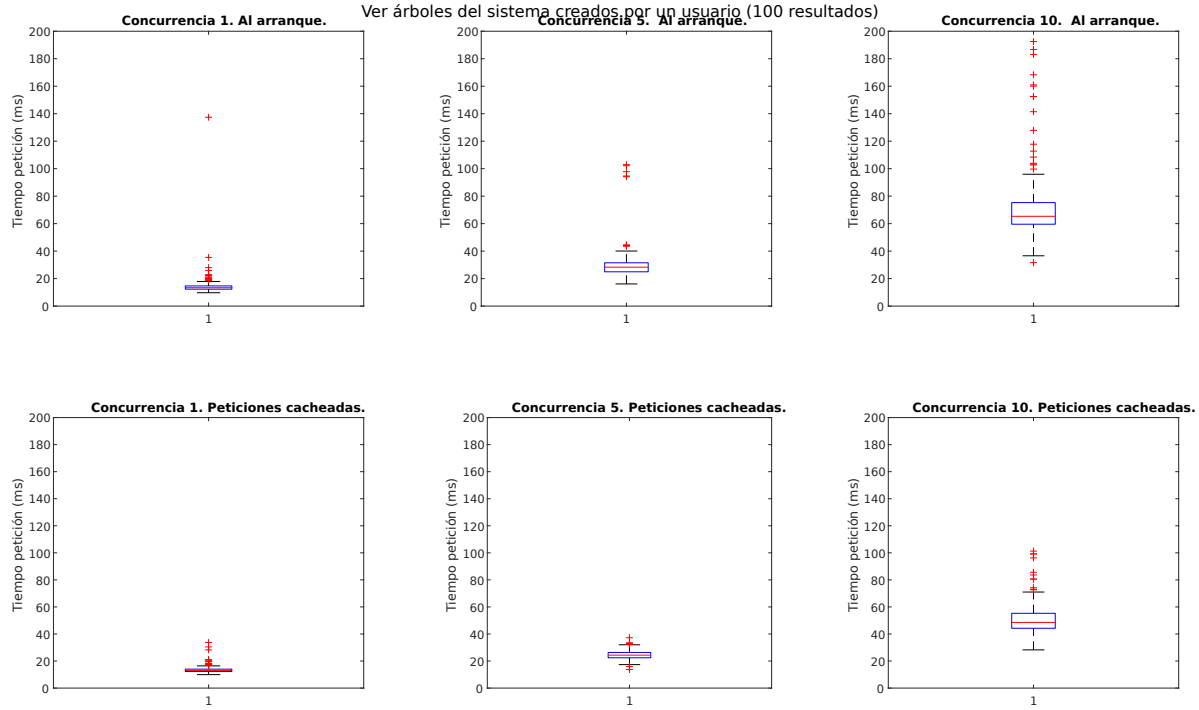


Figura F.8: *Boxplots* de los tiempos de respuesta para peticiones de ver árboles de un usuario con `loadtest` para 30 segundos de experimento.

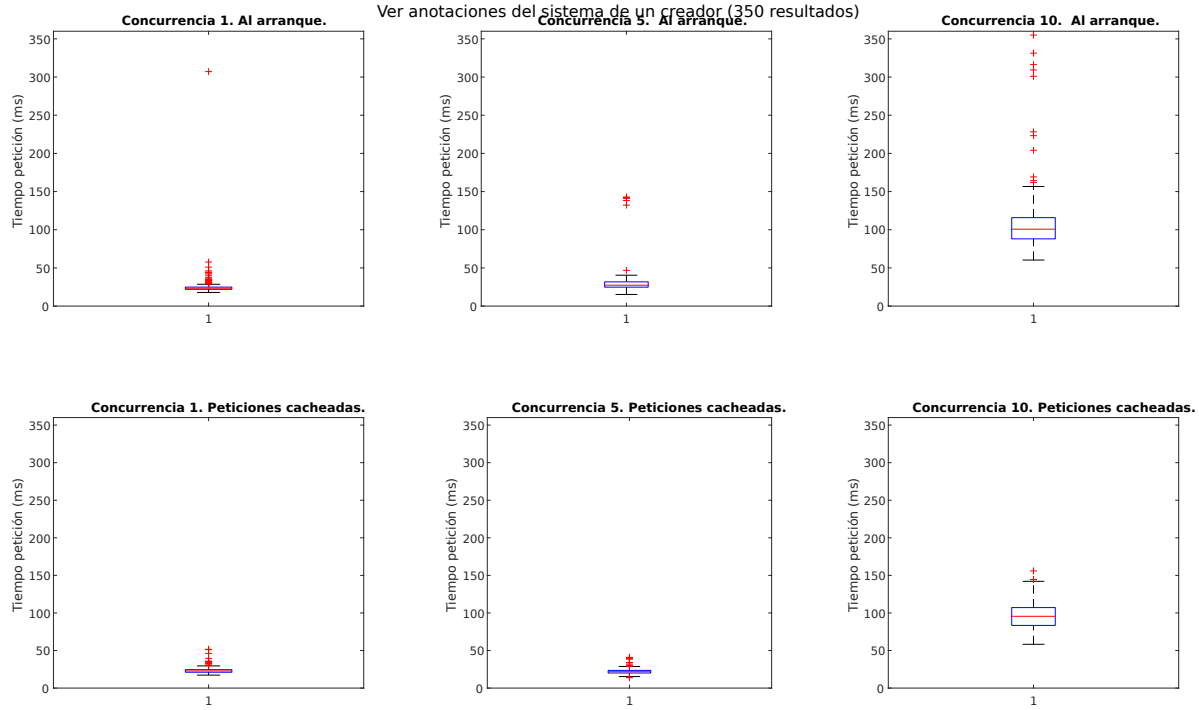


Figura F.9: *Boxplots* de los tiempos de respuesta para peticiones de ver anotaciones de un usuario con `loadtest` para 30 segundos de experimento.